# LINFO2401 - Open Source strategy for software development

Student

Last modification : January 10, 2026

## Contents

# 1 About the course

## 1.1 Why ?

The point of this class is not to add more superficial knowledge, but to encourage reflection on what computers are, what writing code means, and how software affects society. We distinguish **engineers** and **theologians**. Engineers work within non-negotiable physical laws to build things that seem to defy nature, such as bridges or airplanes. Theologians, by contrast, treat arbitrary human rules as unquestionable truths and forbid asking "why."

Traditional engineering disciplines must obey physical constraints (gravity, materials, stability), but most software engineering operates almost entirely under arbitrary human-made rules rather than natural laws. As a result, many software developers function like theologians: they follow established conventions without questioning their origin or purpose and resist critical inquiry.

So, let's try to change that and put back "science" in "computer science" and "engineering" in "software engineering". Let's become engineers. True engineers and scientists are defined not by diplomas or accumulated knowledge, but by their habit of questioning. The theologian asserts "it is so," the scientist asks "why," and the engineer asks "how", a question that necessarily begins with understanding "what" is really going on. [1]

Method of thinking : 1. Write the problem 2. Think hard 3. Write the solution Most problem are because you focus on step 3 (writing the solution first). But what is the problem really ? Always question what you are doing : WHY ?!

## 1.2 Project Guidelines

- **Contribution Strategy:** Analyzing open issues, communication etiquette, and avoiding "diving straight into source code."
- **Storytelling:** Explaining choices and interactions (Focus on human interaction over technical code). Why did you choose this project, did it was your first choice, how did you exchange with people. How is the project managed, decision taken, how are interaction ? What is the project, who is involved. Convinced the teacher you learn something, what is open source and human interaction. Important to contact the community. Just a message, then work alone and a pull request 2 mouths later isn't good. Often their is tickets
- **The Journal Approach:** Importance of writing day-to-day progress.
- Do all git command with the terminal
- Put the link to your contribution in the report.
- The contribution should be merge, or at least the student must have done everything so that the contrition will be merged. Student must follow up why the contribution was not accepted.
- Must thing to do is be able to be able to compile the project. So do not try to contribute to firefox / libre office. You must be able to see your contribution yourself. Compile and do a very small modification.
- **Git Workflow:** Commits, Pull Requests (PRs), and merge requirements.
- **Problem Solving:** The "Why, What, How" method (avoiding the rush to solution).

[1] Section based entirely from Chapter 1 of the syllabus

**Comments about the oral presentation :**

· Teacher first clicks link to see bug report, see the commit, see the interaction with the community. · Not too technical; insist on the interaction with the community. What did you learn from the whole process? · Why you chose this one. Why did you fail? How did you choose the next project? · Before starting, make sure you can build the project on your computer and that you can modify it. The teacher is not judging your code. · He judges how much you learned. · The point is interaction with the community, not just coding alone for 3 months. · Book: What did you learn by reading the book? What links did you make with something else? Don't do a summary of the book.

## 1.3 Expected Exam content

Note that you can come with your own exam question. If you are passionate about something, talk about it, what are the current open-source project of your passion ?

Other that that, you can expect :

- Analyse an Open Source project (License, governance, community, challenge).
- Being able to do basic git commands through the terminal.
- Explaining what is a License + examples.
- Difference Rival good VS non rival good
- Given the description of a License, explain what type of license it is.

## 1.4 Book

You can read a book to get one bonus point.

# 2 Early History

## 2.1 The importance of writing

To understand computers, we should view them primarily as communication devices, not merely as machines for numerical computation. Their origins lie in humanity's fundamental need to communicate across time and space.

The most decisive invention in human history is **writing**, which made history, truth, time, and distance possible concepts. Before writing, knowledge was mutable, time was limited to the present, and distant places were largely mythical. Writing transformed human cognition by enabling communication with absent people, including those long dead, and by providing durable proof of facts. Because of its power, writing was often controlled by educated elites.

Writing also differs fundamentally from speech: it forces clarity and logical consistency, empowering the reader rather than the speaker. Tools that read and write on our behalf weaken our ability to think, and change how we appear for people reading us.

**Solution for faster long distance communication :**

For millennia, writing was limited by human and physical speed, making long-distance communication painfully slow. This limitation drove the search for faster transmission methods. Early solutions like the **semaphore** system relayed visual signals between towers and worked effectively but were costly and labor-intensive.

The major breakthrough came with the **telegraph**, which used electricity to transmit messages over wires. The use of electricity automatically bring a binary mentality (current flow on or off), and led to coding systems such as **Morse code**. Telegraph operators became fluent in this new language, forming social bonds, playing games, and even conducting relationships over long distances.

With the development of reliable submarine cables, the world became globally connected, allowing written messages to travel across continents in hours rather than months, laying the foundation for modern digital communication.

[2]

## 2.2 Typewriters and Keyboards

Christopher Sholes developed the typewriter to produce standardized, readable documents, offering an alternative to slow handwriting and expensive printing processes. Among many competing designs, his machine stood out.

In his prototype, the keyboard were not aligned due to technical constraint and jamming. A temporary fix was to randomize the layout to avoid frequent use of letters next to each other. **This lead to the QWERTY layout**. Even after the defect was fixed, the layout was kept for commercial reasons, to avoid admitting flaws in already sold machines. As a result, an inefficient and arbitrary keyboard arrangement used worldwide.

National variants such as **AZERTY** and **QWERTZ** likely arose not from technical necessity but from commercial protectionism, a trend later reinforced by IBM, which further fragmented layouts across countries (AZERTY BE, AZERTY FR).

Although typewriters symbolized administrative work, an unexpected effect emerged. People (especially professional women typists) learned to type extremely fast. This efficiency led major

[2] Section entirely based from Chapter 2 of the syllabus

writers such as Nietzsche, Steinbeck, and Hemingway to abandon handwriting. The typewriter altered not just their productivity, but their style of writing and thinking.

Attempts to correct QWERTY's inefficiencies, like Dvorak's keyboard layout, proved to be technically superior and enabled record typing speeds. However, they failed commercially because users were already accustomed to QWERTY.

If you are serious about writing on a keyboard, learning to touch type is the best investment you could ever do. Anyone serious about using a computer should spend some type to learn real dactylography. And if you are learning touch typing, learning a layout optimized for your language is probably a very good investment. Some current efficient layout :

- English : Colemak (successor of Dvorak)
- Français : Ergol (successor of Bépo)

[3]

## 2.3 Data Encoding: ASCII to Unicode

Origine de l'ascii : merge entre machine à écrire et télégraphe ? Code en 7 bits. Before it was bodot ? A in ascii is for American. So iits good for language without any accent. Each language has its own version (ISO-XXX) UTF-8 is an extension of ascii. So all ascii char are still compatible. Instead of a fixed length for char, 8 bits for all char.

Toujours considéré une alternative et tout les cas possible : html : <im src = "", alt=""> Accessiblitity is good for everybody

## 2.4 Hardware & Early Machines

Premier ordinateur, machine pour craquer enigma par turing & co Punch card : better way to encode data, rather than manually moving big cables. Holes = 1, not hole = 0. First programming language. Batch processing : separation between computer program and the data ?. Computer do the computation on its own ? Bug : "program not acting has it should have". Peoples didnt understand why the output was wrong. They finally enter inside the computer room. They found a dead bug doing a short circuit.

## 2.5 From Batch Processing to Terminals

When computer become faster, we though we could interact in real time. They added the typewriter. Instead of sending electrical signal to wire, directly to computer. It is the first terminal. Its hardware. After their was magnetic tape Modern computing started with UNIX Assembly : set of instructions the computer understand ? Translator between math formula and assembly equivalent. Then we add a program that do the translator automatically. Birth of FORTRAN, the first computer language.

[3] Section based from Chapter 2 of the syllabus

# 3 The Birth of UNIX

## 3.1 From Multics to UNIX

Most of the time when we interact with a computer, it is idle.MULTICS big project coming from the BELlabs. One nerd alone for a mouth build UNICS (kernel, filesystem, shell (command interpretor), and ?) all in assembly. He Formalize the concept of file, tree like directory, .everything in unix is a file. Invention of B: Before, if a small change in a OS was done, they had to change everything.Then it become C and UNIX.

Main frames compared to Microcomputers : couldn't run unix : atari, comodor. Play games and small programs

## 3.2 The Shell and Pipes

Every command-line process in Linux operates with at least two fundamental data streams: standard input (stdin) and standard output (stdout). A program reads data from stdin and writes its results to stdout. Understanding how to control both stdin and stdout is crucial for effective command-line work.

- `>` : The `>` character is a redirection operator. It intercepts the data heading to `stdout` and sends it to a new destination. If you want to add to a file without erasing its contents, use the `»` operator.
- `<` : Similarly, we can also manage the standard input (`stdin`) stream. By default, a program receives its `stdin` from the keyboard, but we can also use files or the output from other processes as an input source. we use the $<$ operator to redirect stdin. This powerful feature allows you to tell a command to read its input from a file instead of waiting for you to type it on the keyboard.
- `|` : In Linux, the command line becomes incredibly powerful when you start connecting commands. Instead of running one command, saving its output, and then running another, you can create a pipeline to pass data directly between them. The pipe operator |, represented by a vertical bar, is the key to this process. It takes the standard output (stdout) of the command on its left and uses it as the standard input (stdin) for the command on its right.
- `tee` : What if you want to see the output on your screen and save it to a file simultaneously? This is where the tee command comes in. The pipe and tee command in linux is a classic combination for logging and monitoring.

[4]

Shell = asking for a command (bash, zsh). Not the same thing as the terminal!

[4]Thoses explanations comes from https://labex.io/linuxjourney, the *text-fu* section.

# 4 Legal Frameworks, Economics & Licensing

## 4.1 The Economics of Goods

### 4.1.1 Rival vs. Non-Rival Goods

- **Rival :** If i take pasta from the shop, their is one item less in the shop, the shop lose money, only one persone can use it. stealing. A rival good, cannot be used by two people at the same time. Taking it without paying is clearly stealing because it deprives someone else of it. If you take an apple from a shop, it's one apple less for someone else.
- **Non rival :** Non-rival goods can be used by multiple people simultaneously without loss, such as access to a bridge. Using such a good without paying may be illegal or immoral, but it does not constitute stealing in the strict economic sense because no one is deprived of anything. their is a bridge and you had to pay to cross the bridge. If you don't pay and cross the bridge its not stealing in the sense that people who build the bridge didn't lose money (lost money opportunity)

  Coying music from cd to tape was a quality lost but not a problem, bc music was associated with the physical object. Before a book was really expensive (monk copy) then cheap with type machine. At the beginning their was no protection for author. The content could be "stolen", ideas could be spread across everywhere. The idea inside the book was not rival anymore. So it was a problem for the king that wanted to keep their power. Copyright was a pure censorship of book.

## 4.2 Intellectual Property Laws

First time we have seen "intellectual property" : greek city where the inventor of a recipe had to be paid for one year every itme a cook was doing the dish.

### 4.2.1 Software as a Good

Historically, computers themselves were rival goods: they were large, rare, and impossible to steal. Software was initially viewed merely as instructions for using a specific machine, written for one computer and assumed to have no value elsewhere. As software grew more complex, the idea of portability emerged, making it possible to reuse and share programs across machines. UNIX was created and developed that way, most software were developed by sharing and copying. The computer industry was willing to encourage that fact as more software equated with more computers sold.

At that stage, software was not considered an economic good in itself, but more like instructions for using an object like dishwasher. You wouldn't sold the instructions to use the dishwasher you already paid. Programmers freely shared software without commercial intent, because the real business was selling computers. More available software simply increased the incentive to buy hardware.

### 4.2.2 Two Major Systems

Many countries, such as France, rely on "droits d'auteurs", patrimonial rights were the author has recognized perpetual rights on a work.

- **Droit d'auteur (France/Belgium):** composed of:
  - *Droit Moral:* Perpetual, inalienable rights (right to recognition, right to integrity of the work). Example: An author can block their work from being used by a nazi party

because it will make him look like a nazi himself

  - *Droit Patrimonial:* Economic rights (right to sell/copy), simialr to the copyright.Their is no taxes on this rights (at least in Belgium) generally in a contract, you give all your patrimonial rights to the employer.

  Anglo-Saxon systems instead use copyright, a concept that originated in England as a tool for controlling and censoring book printing, and later evolved into a transferable "right to copy" non-rival goods.

- **Copyright (Anglo-Saxon - USA/UK):** Originally a tool for censorship and printing control in England. It evolved into a transferable "right to copy." It focuses more on the economic aspect than the moral rights of the creator.

#### 4.2.3 The Copyright Spectrum

Society often presents a false dichotomy: you are either the absolute owner (Copyright) or you own nothing.

- **Default Copyright:** In our society, every work is, by default, under a copyright licence. Meaning that the buyer can't do anything else other than consulting and using the work for its personal use. Any other use, share or modification is by default prohibited.
- **Public Domain:** Public domain works aren't associated with any particular right: anybody can use, modify and redistribute it as they want.

One of the majors intellectual scams of the copyrights absolutists is to have made us believe that there was no other alternative other than those two extremes. As if we were either owner of the apple, or we not. The fiction wants us to be either the owner of a work (of the copyright), or owner of nothing, just good enough to watch. It is obviously fake.

Licensing operates in the vast space *between* these two extremes.

You can sell a copyright but not a droit d'auteur. So what happens when the author died ? 50 years after death will go to public domain. So every body could use it.The US was really afraid bc mickey mouse will soon reach this date (50 years after death of Walt Disney). He didn't wanted mickey to go to public domains. So mickey mouse laws and become 70 years. Currently it became public domain. So suddenly their was a big hole in the public domain. Catastrophe for the culture bc if the author think his work will not be popular, he will stop publishing. And cannot copy his work bc of this law. So all those works will died and be forgotten. When their is no 1 author, the law is 70 years after the last work. So for Tintin, the guy will publish a new book one year before it reach this date just to extend it.

#### 4.2.4 Patents VS Copyright

Trade mark ? Patent : temporary monopoly on what you have invented. But what about when 2 different people got the same idea independently ? Group that decide the inventor. However you don't have to invent it. You just have to invent it.its basically lawyer discussion to lawyer. So company tries to do has many patent has possible. When patent infringement it will go to "who has the most patent", even if they are all stupid. In the pharma industry, they slightly modify the formula so that their is a new patent .and under the same name.

When independence : USA said that they will not recognize any patent from Europe. So they can copy everything and grow faster. That's how every country start when they want to become industrial giant. WW2 has made american product know therefore they got a market to sell their product like coca cola. Pirate bay was perfectly legal in sweden because it was just text (torrent)

that couldn't be copy righted. Pirate bay fucked around with the copyright noticed until one day. Police raid ? It took everybody by surprise; USA send treat to sweden, if those guys of pirate bay didn't go to jail, USA will stop "obama deal "? Mutual investment of military ? USA impose they view of the world to others. Kopimism religon, life if share by copying religion.so if people went to court you can say you are kopimislm you could copy code. Freedom of relgion won in sweden.

Stuff who "take for granted now" where choice by politics to convice people that their is no alternative.

Piracy is not bad :) the brain washed with the small scene at the beginning "you wouldn't steal a car, piracy is a crime", well the font they use was pirated, and every pirate didn't event see this scene. OpenAI, Meta can pirate to train their AI models just bc they have money. Some are protected by the laws, others must obey the laws.

## 4.3 Licensing

### 4.3.1 License as a Contract

To keep things simple, in our societiy, every exchange follows a certain contract, it can be implicit, but it exists. If one buys an apple at the market, the implicit contract is that after paying for it, the apple is his and he can do whatever he wants with it.

**Definition :** A license is a contract. In this contract, the customer agree to pay some money in exchange for the ability to use the software under certain conditions. The seller may impose conditions to use the software. For example, it could be said that the software cannot be used on a Tuesday. The example is absurd but this could be part of the contract.

A license is not revocable! If a file is distributed once with a permissive license, you can't change it later.

**Dual Licensing :** It is important to signal that each transaction comes with its own contract. It is possible to give rights to a buyer and not to another. It is in fact that principle that allows the "dual licensing" practice.

Where things get tricky is when the exchanged good is said "non-rival". Which means that the good can be copied or bought multiple times without any impact on the buyers. In our case, we typically talk of a software or numeric work (would it be a movie, book, music, ...). It became evident that numeric shopping doesn't grant us any ownership of the work we bought.

It is important to signal that, for long, non rivality of works as music, book or movies was disguised by the fact that the support itself, was a rival good. If I buy a paper book, I'm the owner. But I still don't own the rights on its content! Numeric support and Internet dissipated that confusion between work and support. To regulate all this, buying a numeric work or a computer program is like every purchase: it is subject to a contract that stipulates the exact rights and obligations that the buyer will receive. License is nothing but a contract pattern, a sort of standard contract model. This contract, as for a good part of our society, relies on the supposition that, like for a rival good, a non-rival good needs to have an owner. That is obviously arbitrary and I invite you to question that premise

### 4.3.2 The "Wall of Bricks" Analogy

If the license is a wall of obligations to which the buyer must submit, it is possible to only take some of its bricks. As an example, one can grant all ownership rights except claiming the

paternity of the work. The BSD, MIT or Creative Commons By licenses, for example, require to cite the original author, but we can still modify and redistribute.

An important thing to point out is that when redistributing an existing work, we can modify its license, but only if we add constraints (bricks). So I have the right to take an existing work under CC By license, modify it and then redistribute it under CC By-ND. However, it isn't allowed for me to take bricks and do the other way around. In every redistribution, the new license must be either equivalent, or more restrictive.

The problem of this approach, it that everything will end up restricted because we can only restrain user rights! That's what happened in big companies like Google, Facebook or Apple who use thousands of open source programs and transform them into proprietary software. A real open source patrimony looting!

### 4.3.3 Copyleft - "No More Bricks" brick

That's where Richard Stallman idea stands out: by inventing the GPL lincense, Richard Stallman created the "no more bricks" brick. You can modify and redistribute a software under GPL license, but modification must also be under GPL. It's also the idea of the Share-Alike clause in Creative Commons.

Ironically, "copyleft" refers to licenses that prevent the addition of bricks, and thus the privatization of resources. They were presented as "contaminating" or even as "cancers" by Microsoft, Apple, Google or Facebook. Those companies now present themselves as big defenders of open source. But they fight with all their might against the copyleft and against the adoption of those licenses in the open source world. The idea is to pretend to open source developers that if their software can be privatized, then the companies, the great princes, will use it and eventually...maybe...hire the programmer and pay them a few peanuts.

The truth is as obvious as can be: as long as they can add privating bricks to licenses, those monopolies can keep exploiting common good represented by open source software. They can benefit from an impressive amount of free or cheap workforce. The fact that those morbid monopolies can keep exploiting and are even praised by the exploited developers, shows the fundamental importance to understand what really is a license and the implications of the choice of one license instead of an other.

## 4.4 Types of Software Licenses

We couldn't say who was the author of a given software. 1 line of code could belong to 1 person. So license is like a contract : an agreement between two party. Between the owner of the copyright of the software and the user.and could be a contract for each user. A license is a standardized contract and its not tied to the software but to a specific distribution of the software (bc can vary to each user). For example if we dl soft from github they will be a specific license.

A license is a contract between the owner of the software and the user. The owner of the rights of the software is not necessarily the developer; he can sell the rights (for a salary). They apply to a specific distribution of the software. So for the same software, you can have multiple licenses, dual licensing for the same software. By default: each file has its own license.

**Licensing agreement :** I give my code to the project (I'm still the author, credits, droit d'auteur), but I give up my copyright (droits patrimoniaux). You (the project/company) are free to update the license without informing me. Otherwise, they would have to contact everyone.

### 4.4.1 Weak Copyleft

LGPL is for libraries and does not contaminate the whole process.

### 4.4.2 Strong Copyleft (The GNU Family)

GNU : (gnu is not unix) whole operating system with its compiler gcc + emacs. So the gnu public licence (GPL) to garentee a soft stay libre. So the 5th freedom is " you cant remove freedoms". Concept of copyleft?

GNU/GPL, you can change the license but with similar rights (compatible license). If you put a GPL code fragment, the whole software becomes GPL. GPL2 (Linux kernel license): Tivo problem, where you couldn't run other software; and GPL3: you cannot prevent other source code from running on the same hardware.

The user = the one who runs the code on a computer. When you do a Google search, the user is Google's servers themselves. AGPLv3: the users are the ones using the output? Anyone connecting to the server has the right to get the source code?

Agpl, is copyleft over the network : everyone who use the soft through a network must also be able to get the source code. So it's More than gpl

### 4.4.3 Permissive (The "Do What You Want" Family)

Berkeley : was doing a lot of soft and made BSD. Before their was no official unix so everyone wanted to do its own.

One of the first license was the BSD : you can do whatever you want but you have to list the author.so compatible with the 4 Free statment of Stalman. But we cannot guarantee that the software stay free. The license happens when their is a exchange (dl the source code), the act of distributing. So if you dl a code with a specific license, you have this license for life even if in github the license change.

Some license would soon appear after Richard Stallmann defined the GPL and the 4 freedoms. Those licenses would be granting the access to Free Software but without guaranteeing the freedom that comes with it. Examples include BSD and MIT licenses. The MIT licence allows anyone to use, copy, modify, merge, publish, distribute, sub-licence and/or sell copies of the software. It is provided "as is", users may use the software at their own risk. As a further obligation, all copies or redistributions of the software must contain a copy of the MIT licence. The BSD license is similar to the MIT license. However, the BSD licence may have advertising clauses that require the original authors to be mentioned in any publicity relating to the software. Some BSD licence variants do not contain an advertising clause (FreeBSD)

## 4.5 Non-Software Licenses : Creative Commons

Adapting licenses for non-code assets (BY, SA, NC, ND).

GPL, BSD, etc. are really focused on software. Now we want licenses for anything else: pictures, textbooks, games, anything that is not pure code. They can be used for code but don't fit well.

Games: license for the code and license for the assets of the game. The goal of the CC is to make a set of licenses for your own case. Do you want attribution to you? Do you want to allow other people to use your work commercially? Derivative works (can people modify it)? Are they forced to use the same license?

**Clauses :**

- BY: credit the author
- NC: non-commercial (so not a free license).
- SA: share alike, like copyleft, you have to keep the same rights (not a more restrictive license). A published work under CC By-SA can be modified, redistributed and even sold. The only condition is to still be under CC By-SA license or equivalent.
- ND: no derivatives, I don't want any modification. Redistribution, on the other hand, is allowed.
- CC0: no clauses, do what you want. As an author, you cannot abandon your rights, so you use an explicit license to say that ( like public domain).

CC is on version 4, because there was a bug in version 3. The license applies if you respect all of the clauses. If you don't give proper credit, the whole license is revoked. If you use a picture and don't credit it, you don't have the rights of CC. The default license applies, and it's standard copyright. So many scammers created databases of pictures with very long names. When people used the pictures and made a mistake in the name, they received cease and desist letters and were asked for thousands of dollars.

# 5 Proprietary VS Free Software

## 5.1 The Rise of Proprietary Software : Microsoft

In Section 4.1.1 we explained the at the beginning, software were freely shared. In 1976, Bill Gates tried to fight this philosophy by instituting software as a real business. The concept of License emerged to prohibit peoples to copy and share a software they have bought once.

Only a handful of developers were considering the strange idea that code should not be copied. Amongst them a young programmer called Bill Gates. Unlike many, Bill Gates had only a business vision of the software world. His father, a famous and rich lawyer, probably transmitted him his legalese and business ideology. With his partner Paul Allen, Bill Gates had developed a BASIC interpreter and was selling it through their company called Micro-Soft (two words at the time). It should be noted that they hired multiple programmers to help them.

Bill Gates wrote "An Open Letter to Hobbyists" where he claimed that developing software cost time, money and that people should not steal software. This Open Letter to Hobbyists is an important landmark because it illustrates how common copying and sharing were. It also illustrates that a business oriented minority was trying to advertise a new ideology in which "sharing is stealing". Spoiler alert: they managed to make that ideology so successful that the "poor" Bill Gates became the richest man on earth only twenty years later

1976 : ad in magasing to the company micro-soft. It said ouin ouin soft cant be use for free bc need to pay the dev. However noone cared about it. Everyone shared everything. The spirit of the day was, if you pay for a soft, you could easily copy it and share it to your friend. Bill gate tried to change the world and make it proprietary. Intellectual proprietary was not really defined Lisp : ancestor of AI. People was exchanging

The “Letter to Hobbyists” and the shift to paid software (BASIC, MS-DOS).

Before, the bigger the computer, the best. Nobody beleived a small computer would play a role. Small personal computer for hobbyist (game). Each brand was sellinng their own comuter with their own OS. Someone at IBM had the idea to sell parts of the computer and sell them. Called “personnal computer”. IBM was focused on “mainframe” big computer. At the end the didn’t had any operating system ( OS/2 not finished and CP-M fails).a stock lawyer of IBM had A rich friend ? Told it had a son that works in computer and could do a operating system. So bill gates went IBM and works with it. Software called “micro-soft”. However it was not really an OS. -> Q-DOS (quick dirty OS) -> MS-DOS. Very shitty OS, 1single process, no GUI. They didn’t sell the OS, they tell to get some money for all pc sold with it. And they tell ibm to be oblifated for each soft had to use their soft. They accepted. At the end it was the only OS. Later new company came to make compatible computer part (hp, dell, ..) Everyone was trying to use their own version of linux (xenix by microsoft, hpux by hp, solaris by ?) so hard to make software that will run on all computerter The only reason windows is so popular, is bc it was what was in all pc, so dev would do software for windows so if you wanted to get the software, you had to use windows. Still the case today, the only reason we use windows is bcause soft are made for this bc everyone use windows bc sold with each pc.

## 5.2 AT&T’s Monopoly

How antitrust laws forced UNIX to be shared initially, then closed later.

Because AT&T was consider a monopoly, they couldn’t make use of UNIX so they let the searcher do whatever they want. SO they were sharing code with university. So at the beginning it was all open source. Everyone has its own version of unix. Open source and sharing knowledge

(pirating) is the default

When monopoly where “good for the gouvernement”, Unix “belongs “ to At&t. BSD was probably illegal so everyone afrait to use it.

AT&T was afraid to use UNIX bc monopoly was a really bad thing. Then AT&T say it must become “closed source”. So researcher “forgot” the tape of UNIX so that it still can be shared. Student weren’t allowded to see the source code... Birth of BSD

### 5.3 Apple VS Franklin

The legal precedent establishing that binaries/ROMs can be copyrighted (software as property).

Visical was the first spreadsheet soft, running an a apple software. So everyshop wanted to buy an apple computer bc of this soft. -> dump of the rom from apple computer to own rom to run visicalc. (apple computer vs Franklin computers). Apple sued, but the defence said they just copied 1’s and 0’s. This is machine output so cant be copy righted. Sundenly their is a law forbidding copying binaries. In 7 years, proprietary didn’t existe, bill gates crying about not to copy his soft, to impossible to do it USA impose their laws to others bc they are really powerful. The pirate bay : in sweden it was legal to dl and share movies. Sweden changes its law bc was menacé de rompre ses deals militaires avec les USA? Pricipe of science : exchange information. Innovation comes from inspiration

Apple won appeal against franklin (binary code)-> politics favor corporation. Company wouldn’t be able to grow if others could use them. It was a choice to give more power to corporation so that they can grow.

### 5.4 The Reaction: The Free Software Movement

In 1980, the United States Congress added computer software to the copyright code. In 1983, it was clarified that a computer software could be considered as literary work and thus could not be copied or modified without the author’s consent. At that point, software vendors explained that they never sold their software but only an utilisation license, exactly like an editor doesn’t sell you the content of the book but only the right to read it. One of the most popular condition that quickly arose was that users were prevented to copy the software, to modify it or to distribute it. Software became proprietary and, for the first time in the history of computing, people started to use software they could not modify or copy.

**Richard Stallman :**

a programmer called Richard Stallman was happily hacking in the MIT artificial intelligence lab where he started in 1971. "Hacking" was the word they used at MIT to describe "solving a hard problem". The problem itself didn’t have to be serious. The spirit at the time was to make complex practical joke. Everything in life had to be considered as a problem to be solved with the most original solution. Sharing ideas was a strong part of that culture. As computer ressources were rare and expensive, it was seen as counter-productive to keep a computer sleeping. If a computer was locked in a room for the night, Richard Stallman and his fellow coworkers made it a mission to free it in order to run programs on it. Freeing computers involved lock picking, crawling in the ceilings. Having fun.

Richard Stallman : working at MIT. Complex jokes (fire truck on roof). Richard wanted to see a code of a printer and get refused because “proprietary”. Richard took the source code to the computer controlling the printer that get jammed. He modified the code to send email when the print worked. MIT called him an “hacker”. Then they got a new printer with the same problem. He wanted to see the source code but couldn’t find it. Send a message to the vendor but they

didn't really wanted to send him. But he had a friend working their who wrote the code. He asked him but was not allowed to share. Couldn't believe it bc he lived in a worl where everyone could share everything. Same thing happens at the MIT, new vendor arrived. The code he wrote was used and extended and make proprietary. Richard not happy, re implemented all the features for free. That was crazy he could catch up alone for some time.

One day, the printer was replaced in the MIT department where Richard worked. Printers were huge machines prone to being blocked. Richard had, long ago, modified the software of the printer to send a warning to the person requesting a print if the printer was blocked. As not everybody was working on the same story, this simple change prevented a lot of unnecessary walk through the MIT's corridors. Richard knew that the Xerox company would not send him the sourcecode to allow modifying the new printer. But he knew someone who worked on it and though he would simply ask him as he was visiting his university. Richard asked the source code and was expecting either a "yes, of course" or a "no". To his astonishment, he received a third answer: "I can't" At that point, Richard Stallman understood that even good hacker and programmer could be forced to not share their work through contracts and licenses.

When he was refused the code to the new printer, Richard Stallman understood that computer users would be divided into two groups: programmers and users. Users that would not understand anything because they could not, even if they wanted to. He also understood that programmers were only programmers inside their own little company, they would become powerless users of all other software. They could not cooperate anymore. They could not share anything. They would become lonely.

**Definition of Free Software:** The 4 Freedoms (Use, Study, Share, Modify).

He could not fight back the fact that software could be copyrighted. But he knew that a license was a contract. And that you could always write your own contract. Therefore, he imagined a contract that would give you the freedom to modify and redistribute a software. In fact, he even managed to write that contract in a way that would guarantee that the software stays "free". The word "free" should be understood as in "freedom", not as in "free beer". That's why the french/spanish world "libre" is often used to clarify the meaning. FLOSS means : "Free and Libre Open Source Software". But "Free Software" was often confused with "Freeware", softwares that are proprietary but distributed freely (often as a limited version of a paying proprietary software). People building Free Software were looking for a clarification. Some hackers managed to find the word "Open Source".

This contract was called the **General Public License (GPL)**, and software that respected the freedom of the users were called "Free Software".

Richard came with 4 free of software. Free as freedom, not like gratis. Those are just arbitrary but for him, a soft is free if : 1. Use : The right to use it how you want it 2. Study : The right to study it (requiring the source code). What was the decision was used in the software. So you need the source code 3. Modify : The right to modify it 4. Share : The right to redistribute it (you can make copy of it)

If you don't have those four freedoms, you will quickly hit arbitrary limitations. Those four freedoms are only a clever way to explicit the "right to use the software with full control over it".

The GPL not only provided those 4 freedoms. It also ensured that those freedoms would be preserved, by stating explicitely in the conditions of the contract: should the user redistribute the software to someone else, he should provide those same freedoms to that other person. This concept is named "copyleft", where the only restriction you have to agree to is... to not create

any restrictions. You have all the freedoms of the software, except the freedom of taking away some of these freedoms from others. You cannot use a GPL code in a proprietary software. If you do, the proprietary software would automatically be "contaminated" and become GPL. Other licenses like MIT and BSD will soon appears.

It is important to note that **nothing prevents anyone from selling free software**. As long as you give those same rights to your customers, you can sell the software. Richard Stallman himself earned a living for multiple years by selling copy of Emacs, at a time where Internet connection was rare. You would send him money and an envelope, and he would send you a disk with the software.

Richard left mit and created emacs, started to sell it : you send a tape with a check and copied to source code to the tape and send it back. He encouraged every one to copy and shared it.

Eric Raymond and Bruce Perens coined the term Open Source hoping that this would convince the industry that Free Software was not only for moneyless kids and punks. There was a real business behind it. From the start, Richard Stallman was really worried that the word "open source" would hide the fact that Free Software were about freedoms, not about technicalities. By hiding the word "free", we would loose the freedoms. In insight, he was perfectly right. Eric Raymond was famous for writing an essay called "The cathedral and the bazaar" in which he defended that software was so complex that you could not make it with only a few persons. That making it open source forced to write good software, that bugs would eventually be found, that people would contribute brilliant unexpected patches.

## 5.5 Pros and Cons of Propriatary / Open Source

· Security by obscurity: completely wrong — when coding in open source, you're forced to care about it. · Package the software the way the customer wants. · Business model: becoming a big company by distributing and packaging free code. · Maintenance, security, support → huge work, which is not writing code. · Long-term sustainability: company switching priorities, company going bankrupt? · Code quality: Proprietary — the product matters, not the code. Most of the time, it's poor-quality code.

Proprietary doesn't means its good, far from it. Teacher example : In toyota car, they put the worst engineer fo rthe soft part. "bc a car is about mechanical". Toyota missile and they claimed a mechanical failure. But after investigation they found the worst code ever.

# 6 The Linux Revolution & Modern Development

## 6.1 The UNIX Wars & Standards

happened before or during the birth of Linux. It explains why Linus created Linux (because BSD was in a lawsuit).

**BSD vs. System V:** The lawsuit that slowed down BSD adoption.

BSD vs. UNIX: Bell Labs sued Berkeley, saying BSD used codes from UNIX. Everyone was afraid to use BSD because the use of BSD could become illegal if they loose in court. But GNU/Linux appeared with the web. In fact only 6 files were from Bell Labs, but it took years to find them. Once they rewrote them, BSD became officially okay to use.

**POSIX:** The attempt to standardize UNIX (Interfaces, Shell vs. Terminal).

There was not one UNIX release. Source code was just exchanged, and you adapted it to your needs. Now the world has changed; source code was made by companies for people who didn't care about the source code. Companies made the source code proprietary. Open source was the default.

UNIX was very fragmented. One software couldn't run on another version. The idea of POSIX is to define a standard. To create a standard, you need deep knowledge from various experts. When the group agrees, they release a standard. There is no obligation to follow a standard; it's a choice to follow them (also a marketing practice to say you follow one). If you want to follow one, someone has to validate that you correctly follow it. For W3C, there are also automatic tools to check if your website correctly follows it. Companies could follow a standard just before they are audited, then stop following it but still get certified. Organizations for standards: ISO, OASIS, IEEE, W3C. The problem with all UNIX versions is that users were captive under one version. POSIX defines standards to explain what UNIX is. It even defines what tools you should have (ls, cd) and their arguments. That's part of the POSIX specification. GNU by Richard Stallman followed most of the POSIX specs. He also added more features. The GNU extension was to add more verbose options for arguments: ls –all compared to ls -a. All UNIX tools were bundled in BSD.

## 6.2 The Missing Kernel

Linus Torvalds wanted to use UNIX but for microcomputers (small personal computers). At first he just wanted to be able to connect to a UNIX computer. He was not satisfied with the OS he had. He built an assembly software to automatically start a session to the university UNIX machine, just to access the "usenet group"? and talk to people. Usenetgroup was like a forum (no web). He showed it to his friends and added multiprocessing to his computer (in assembly). When pressing Ctrl+Alt+F2, a new terminal popped up.

One day on Usenet, he posted a message about Minix (UNIX for students to learn). He said he had made a small OS.

Linus discussed on the forum minix on Usenet.mini unix for learning. People was interrested of linus works and started doing contribution. There were not contribution allowed on minix. So linux got new versions.

**GNU Tools + Linux Kernel = GNU/Linux :** Since Linus used GNU tools, it became popular? He added more and more features until it became a full OS. Most of the tools are from the GNU project, so we say "GNU/Linux." He was greatly grateful to Richard Stallman's tools, so he put his OS under GPLv2.

GNU –> implement unix soft with copyleft license. (copyleft : If you don't distribute it, you dont have to share the code, its a contract between code author and the user. So if no user, no need to make public. User have the right to share it themself because copyleft means no restriction. Free to give it to someone else. Therefore you become the distributor. )

### 6.3 Evolution of Version Control

Linus started to implement the whole POSIX standard. Linus asked a friend to get space on the university FTP server. This friend named the folder "Linux." Suddenly, there was a new OS you could run on a 386 personal computer (instead of DOS or Windows). Sending a patch (Perl script file) to Linus = a diff between files to apply the patch to the source. If there was no clear conflict, the patch would be applied. He was doing all the work to "merge" it himself, reply, etc. At some point, he applied a batch of patches and decided to release it and put it on the server. The point was not to make a stable release but a consistent one so that people could test it. You could simply send patches by email with Git. Within the email, you could read it, apply it, and make comments. No need to change to a confusing interface. The whole workflow could be used without GitHub (interface).

Linus needed a version control system. The problem was that BitKeeper was proprietary. People made a free client for it, but the company sued those developers. Linus was annoyed about all those discussions and problems. He disappeared for 2 weeks, and he created Git. Git is basically a filesystem. Git stores a lot of diffs. Git explores the folder, recreates it based on the whole history. With Git, you have the whole history, so you can fork it. So even if Linus didn't accept a patch, you could make your own forked version of Linux. This is thanks to the GNU license and Git. Linus Torvalds became the main maintainer. For each part of the source code, there were sub-maintainers who did the first filtering.

By default, a maintainer should say no. Once you accept it, it's there forever. If there is a problem, bug, or security vulnerability, the original person who submitted it will probably disappear in the future. So the maintainer has to understand the patch. Is it something good for the user base to accept the patch? → Start a discussion. Sometimes people write code without realizing they don't really need it. Splitting the patch into small pieces helps the maintainer understand it more easily, and you can realize that some parts might be useless. · The normal way: create patch, apply patch. · GitHub: proprietary, creates a centralized request, creates a merge request, accepts them on the web interface. "Not the true Git way."

Linux kernel mailing list : discussion started their. Linus split the code in separated modules. Over teh time each modules got their maintainer. So each contribution go firest to those sub-maintainer to do the filtering, then in go to linus. Before contribution was rely centrilise. Merge before commit. CVS / SVN was difficult to work on so he he used a proprietary soft. Enough of discussion, 2 weeks of holliday, he developed git. Git his written like a file system. Git was deve for linux kernel dev. Instead of taking all the upstream stages, you can commit loccaly and then you send your merge / PR. Scott Sharpan added a web interface on top of git? Github force to use UI for merge ? Github acquired by microsoft so back to the original problem.

### 6.4 Community Dynamics

Code of conduct imortant, if not written it's implicit (and follows the leader) You can' accept every opinion (framework laptop errors, gave money to withe suppremacist and said they want to accept anyone) With git you can track how the code of conduct has changed.

If disagree with maintainer, you can create your own fork to implement it as you want. Either a simple featuer sometimes it can out lead the original project. (example : OpenOffice was the

open version of a proprietary office suite. However the company started shitting on it, so the community created the fork : LibreOffice. Sometimes a fork can be merged back

Bus factor: how many people does it nned to be ride over by a bus for nobody could maintain / work on the project. When we write something, think about the othres who will need to also work on the same project, or even for you later. If it was difficult for you, write docs to make it easier for others.

# 7 The Internet Age & The Browser Wars

## 7.1 The Networking Revolution

To communicate witha computer, you need to be connected to it directly. Pour connecter 2 ordinateurs : mettre le téléphone sur l'ordinateur. Le pc convertit les signal sonore en bits ? Modem ? What if each computer had an adress and we can send packets to the correct computer. So need to be connected to only one comuter that will route the packets. TCP/IP Web is one application / protocol running of top of TCP/IP (internet). So for now we have internet bt not yet the web. Telnet : before SSH and not encrypted FTP : transfert file SMTP : mail UUCP : unix to unix copy, used by usednet (public forum). Big database synchronyse to all computer. So when you wanted to see a message, you had to dl everything. This is how people started to exchange on the internet

## 7.2 The Web (WWW)

WWW was created by a belgian and a ? Located in france but a swiss office. It was something for the CERN. Present his invention www at the hypertext conference. No one cared at the use bc there was gopher competitor to organise information. Protocols above TCP/iP that are not part of the internet : · Smtp / mime · Ftp / binary · UCP / usenet · Telnet / The only thing missing was how to find stuf. You could dll, communicate, send command, but hot to find and organise ? -> gopher/gophermap appeard. There were no hypertext to jumpto other file. Rober and ? Invented HTTP / HTML because gopher not popular in europoe. They developed the first web browser. Gopher was really intellectual. Mosaic ? University project. At first Tim Berwlly not wanted image inclusion in a webpage. So the guy left his unif and created his browser netscap and invented the hgml tag <img> aginast the will of tim. This Browser was popular, images makes the www popular but become unreadable.

Browser war between netscape and internet explorer. Microsoft didn't interpret html the same way of netscape to mess with their proprietary bullshit. However you had to dl netscape. And netscape used java/javascript for extension ? Yahoo : One html webpage with link to other. Ads was asked merchent on the street if they wanted a add on the site for a day. Netscape bubble went bankrupt becasue the bubble crashed ?

## 7.3 The Dotcom Bubble & The Browser Wars

In the stock marcket you can buy shares and sell it. You buy share to expect dividend to the sock holders. -> old way of thinkingNow you expect that the share of the company will increase in value and sell them back.If only one share was sold to 1000€, and if their is 1,000,000 shares, media says the company is valued at 1 billion €. But it doesn't make sense bc it expects all shares to being sold at once at this highest sold share. A bubble is at some point the price become irrational and it doesn't make sense to buy. Until at some point someone said, "its really irrational, i start ot sell instead of buying", then everyone start to sell and the ppirce crash down. In early 2000's, there was a huge tech bubble.Peoples wase saying taht : " Every body will user compuer, use internet, buy stuff only". Netscape was the main browsser and distributed freely. It has a lot of investors bc people thought it will become a good company. In 2000 there was a huge crash bc people was not buying anymore. The company goes bankrupt, the coode become open source and gave to a fondation company : "mozzila". Later firefox after phenix born, so its the child of netscape. Most company that was selling stuff online goes bankcrupt exept a small comapny selling books, named amzon bc the parents of the founder was really rich. It took thme 7 years to goe to evaluation before 2000 tech bubble explode. However the bubble created a lot of benefits since all the money was also used for the infrastructre of the internet (like cables, ...)

# 8 Privacy & Surveillance

## 8.1 Surveillance Capitalism

Google : Software to crawl automatically the webpage and make a search engine, also nearly bankrupt. Since they have so many information in the logs of the user. They used it to do targeted adversisement. Quickly the amercan departement said its against the constitution to spy.

After the crash nobody wanted to advertized anymore on th einternet. Google was based on advertizing.For the internet userwas still using internet and most ws using google. Every logs of google was quite complete like IP, user agents, .. So they can identify usrs. Then they will sell that to advertizers saying they know the markets and what user will want. In fact google didn't know if it was working and we still don't know. Currently we are in a bubble of personalized advertisement. Privacy : their is stuff you don't want ot share with everbody. Google starts to invades a lot in people privacy, like when you are doing search for illness. You don't want google to record and sell those data. those information could be used against you (china, trump). Tracking peoples localisation of young woman when they go to place were abortion is legal and were procecuded for trying to do it. 11//9/20001 change a lot politically. George Bush to use that event to get political control. All americans were terrified against terrorist. Us administration reallized that all the data about the terrorist they had it. They had all the data and could have realized an attoack could occure. However they didn't had the technology to make those eveidence spot out. So they forget all charges on google for spying on us citizen, and ask google to help the gouvernement spy on citizen. Google become part of the us defence.

Edward snoden was working for the us intellignece. He was really partirote, wanted to become a solded but was too weak. But he was a geek connected to the internet. So he become a spy and was train for that. He saw that the internent traffic was analyzed and strored. After snowden reveal, in fact, nobody cared. Now it's the norm for people to say "it's ok " Once something is public, you can't take it back in private : barabare ? Effect. Reasearcher of coast errosion took picture of the coast and there was a house of barbara. She said it was privacy invasion -> lawer -> complaint and case -> took people attention. From nobody care of her house to everyone knows about her.

The founder of whastsapp was really into privacy (bc knew what it was to be spyed on). He tryid to make sure the content was encrypted and couldn't be use easily. Then he sell whatsapp , against all his principals, to facebook. He had to work for facebook for some years to get all the money but leave earlier bc he was really angry of what they did. Facebook was linking all the data of who youre talking to, the rate, when, ... And they merge data from whatsapp and facebook.

When someone knows he is watched, his behavior will change completely. They comply and are less creative / revolutionary (experiment with a guard in a room). Targeted ads may not work for selling stuff, but for politics it's different. Gerrymandering: very strange way to vote in politics. There are some states / counties / sub-counties where you already know that Republicans or Democrats will win. So there is no point in investing campain money there. There are only a few battleground states / counties. Cambridge Analytica knew exactly those places, who those people were and exactly what the political position of people was. Trump paid Cambridge Analytica and they used data from Facebook. GDPR tries to protect the privacy of people. Peoples has to know when they are spied on, so companies need the consent of users, so they created really annoying cookies and made people think it's because of GDPR. It's false. Marketing people are evil and tried to conveice peoples gdpr is annoying so it should be removed. In fact marketing peoples are affraid of gdpr. No privacy allows others to manipulate people and elections. If you

have nothing to hide right now, you can't say the same thing in 10 years. You don't know who will be in power. If you want to arrest someone but you don't have a reason yet, you just have to look into all the data gathered over all the years and you will find something (even if it's a stupid 10 years old thing). Data from Europeans is more expensive, either more difficult ot obtain or more risky to use due to gdpr. How to protect your privacy: – Fight with law, at the political level. It's worth breaking the law for companies if the fine is less than the amount of money you get from breaking the law. In the VW pollution scandal, it was the engineer who wrote the code who went to prison, not the person who asked the feature to spoof pollution. – Cryptography. To really work it should be open source. It's really hard to do it. Kruger effect: the less you know, the more confident you are about the topic. Never trust someone who claims to have invented a new crypto algorithm. It means he is not able to check that he is himself illiterate on the topic. Telegram by default not encrypted + encryption done by random dudes not proved to be secure. – Not having your data on the company → decentralization or distributed instead of centralization. Email, for example, can be decentralized; Fediverse (Mastodon). Anyway, you need a protocol for decentralization because you need all nodes to agree. But it's difficult to make a standard. You can't force someone to use it; otherwise, it will be centralization.

# 9 Computer Security

**Security by Obscurity:** Why it fails.

Seucrity = Action taken by a community to ensure the members of the community respects the rules of the community. Rules are expected to changes, arebytrary, unfaire. It's about specifc set of rules. The security is bout respecting rules. There is 3 way to enforce security. 1) The moral way : convince people to just follow the rules. The community will try to tell you what is good or bad. The security must based arount your own morality. 2) Cost : expensive in time, money, effort ... To breack the rules 3) The consequeces : Proba to get caught x the concequences. If your security ins't at least one of those, it's not security. Putting military in the street without armed weapons is just a "security theater". A terrorist that will blow up will not be blocked in any by this military. It's just a political stuff to say " we did something". Aireport security is pure theater. 21 out of 25 tester were able to sneck weapons inside a plane during a test. No one knows why liquids are not allowed, bc it doesn't make sense (it doesn't stop liquid eplosion with an quantity restriction) Sometimes security measures can be counter productive. after sep 11, big doors was put for the cockpit. They could be open only by the inside. In 2015 a co-pilot wantend to suicide himself, he locked himself in the cockpit. So we can say that this security measures killed 215 peoples. It allowed another kind of thread. When you do a security door, you assume there will be no danger from the inside. If you don't understand why it's secure, assumes its not secured. Security is about enforcing some rules. Adroid phones were at first open sources. Gradually google removes part of android and put proprietary google services. GrapheneOs is a fork of android to make it secure. Every appsruns in container withrestrictec permission. Google apps are NOT priviledged. Google says "your are protected", but in fact you are not protected against google./e/os cant communicate with googles at all, and it's ore user friendly. Both don't adress the same public Security is always the security of the weakest link : front door with lot of locks but bakc garage dorr open. Security by obscurity doesn't work. When it comes to software, you have all the time you want ot test the soft, so no cost. No consequences when you still test, no moral. Not having the source code just made the cost a bit higher.the best seuciryt soft are all opensource. Bc the best peoples are looking at the sources code. The best secured openrating system is Open BSD. It's 100% open source.minimalistic os, every thing is discussed a lot. Bluetooth was removed bc no one knows how wthe stack was working. A physical key, is security by obscurity. All private partes in NYC were open by the same simple key, one was lost. A journal paper post artical saying "this is the key, if someone

find it, he could opoen everything" well they just realeased the "plan" fo the key so anyone could now opoen each dorr. Snake oil ? People saying i made a system, i cannot find a way to break it so its secured.