Divagations autour du logiciel libre....

Basile STARYNKEVITCH
http://starynkevitch.net/Basile/
basile@starynkevitch.net

à titre personnel !

28 mars 2015, Palaiseau, ENSTA,
pour *Ingénieurs sans Frontières*

Attention ! **opinions personnelles !**

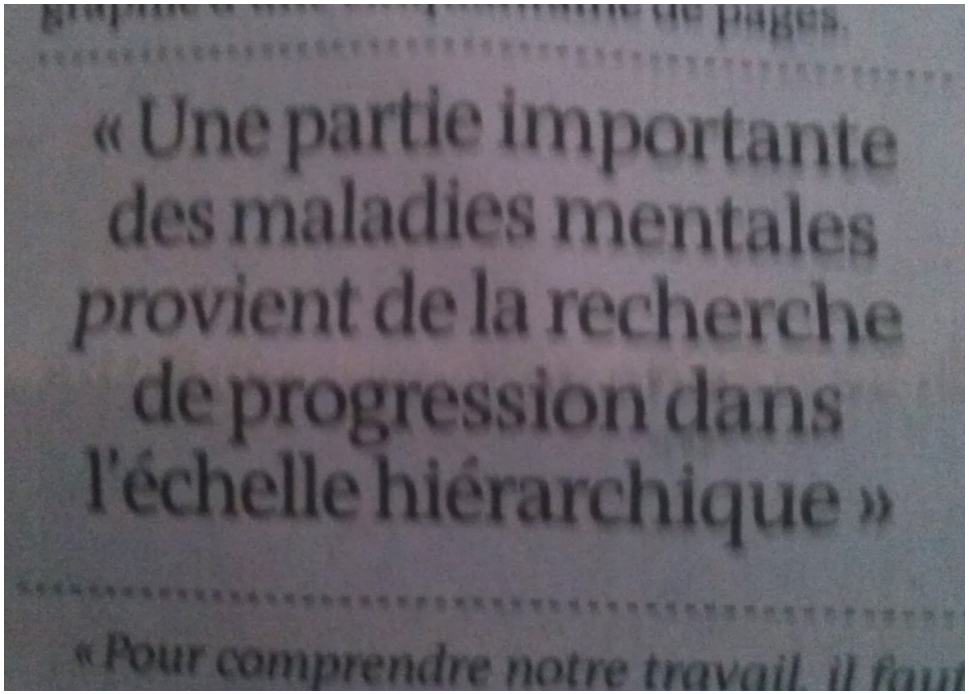
Je (Basile Starynkevitch) suis

- né en 1959 (en France) ; **développeur** depuis 1974 (cartes perforées)
- de formation scientifique : ENS Cachan (math) ; puis thèse en informatique (intelligence artificielle symbolique et méta-programmation)
- marié, 4 enfants, grand-père 6 fois
- membre (trop peu actif) de l'**APRIL** [1]
- contributeur à **GCC** [2] de la **FSF** [3], architecte de GCC-**MELT** [4] (**logiciels libres** GPLv3)
- employé par le CEA, LIST comme ingénieur-chercheur
- mais je ne connais *Ingénieur Sans Frontières* que de nom !
- je ne connais pas et **je n'ai jamais utilisé Windows**

Je ne represente personne et toutes **mes opinions ici sont strictement personnelles** (et volontiers provocatrices, on est samedi)

1. `http://april.org/`
2. `http://gcc.gnu.org/`
3. `http://fsf.org/`
4. `http://gcc-melt.org/`

à propos de management (1)



"Une partie importante des maladies mentales provient de la recherche de progression dans l'échelle hiérarchique."

Tiré d'un article d'un sociologue dans *Le Monde*, novembre 2013

à propos de management (2)

D'après mon souvenir,

Fred BROOKS aurait remarqué :

Si une femme met 9 mois à faire un bébé, 9 femmes ne mettront pas un mois à faire un bébé...

(cette remarque de bon sens s'applique aussi au développement du logiciel et à celui de nos systèmes industriels complexes)

The Mythical Man-Month : Essays on Software Enginneering (1975) ;
suivi de
No Silver Bullet - Essence and Accidents of Software Engineering (1986)

à propos de management (3)

Divagations très personnelles, *provocatrices et subjectives*

- **L'espèce humaine n'a pas un cerveau programmé pour le management** (par[5] l'évolution, ou Dame Nature, ou le Bon Dieu . . .) mais pour la chasse, la cueillette, la vie en tribu préhistoriques de quelques douzaines d'individus (pas en grosse société),
- **Le management pourrait être *toxique*** (psychiquement) **pour ceux qui l'exercent** longtemps[6]
- les 4 dimensions *divergentes* d'un projet : délais, coût (ou consommation de ressources), performances, qualité. Compromis et arbitrage nécessaires.
- **Un ingénieur n'est pas *obligé* de devenir "manager"**
  (il pourrait en devenir malheureux et aurait du mal à quitter des fonctions de management)
- **Principe de Peter** (1968) : dans une entreprise, tout employé tend à être promu jusqu'à son niveau d'imcompétence (le point fixe est l'incompétence généralisée).

Le logiciel libre est probablement aussi une façon de gérer le développement logiciel, et pourrait s'adapter à d'autres sciences de l'ingénieur (Open Hardware ?).

5. au choix, selon vos convictions
6. Le "panagement" via SAP, Excel et PowerPoint est toxique pour la société humaine. Principe de précaution vs Principe d'innovation

La loi (méta-circulaire) de D.Hofstadter

Hofstadter's Law

It always takes longer than you expect, even when you take into account Hofstadter's Law

> *« Il faut toujours plus de temps que prévu, même en tenant compte de la Loi de Hofstadter. »*

Excellent livre "poétique" **Gödel, Escher, Bach** : An Eternal Golden Braid
(Gödel, Escher, Bach : Les Brins d'une Guirlande Éternelle) 1979

---

notions d'**homo-iconicité**, de réflexivité (ou **auto-référence**), de **méta-circularité**, de **méta-connaissances**. Travaux en **intelligence artificielle** de J.Pitrat. blog **`http://bootstrappingartificialintelligence.fr/`**

Matériel

Le matériel (*"hardware"* en anglais) :

- **ordinateurs** : microprocesseurs (4 cœurs à 3GHz) et mémoire (8Go de RAM)
- nos tablettes et **téléphones** (et l'infrastructure téléphonique)
- les **datacenters** (millions de **serveurs** chez Google, Amazon, Microsoft, OVH, . . .) qu'on utilise via **Internet** (qui consommeraient près de **10%** de l'éléctricité produite) et le **Cloud**
- presque tous les **produits** avec de **l'électronique** (voiture, métro, electro-ménager, appareil photo, reveil-matin, avion, imprimante, disque dur, carte bancaire . . .)
- **importance de l'électronique embarquée** numérique et des **micro-contrôleurs** (on en fabrique près de dix milliards par an)

"Loi" empirique de Moore : le nombre de transistors par puce double tous les deux ans (3 à 10 milliards de transistors à 20 nm dans les microprocesseurs, mais on atteint bientôt les limites physiques) ;
Anti-loi : le coût d'une usine de circuits intégrés double aussi à la même vitesse (actuellement on approche dix milliards d'euros)

Le logiciel

- C'est **le logiciel qui fait fonctionner nos processeurs** donc tous nos dispositifs avec de l'électronique numérique
  "briquer" son téléphone c'est "abîmer" son logiciel, pas ses circuits électroniques !
- Les "plans" d'un processeur "sont" aussi du "logiciel" (en VHDL)
- **40%** à 50% **de la valeur d'un avion de ligne** serait du logiciel !
- **le logiciel est la collection** organisée **d'instructions qui est executée par un ordinateur** (ou micro-contrôleur)
- depuis 2012 ? la **majorité des développeurs** de logiciel travailleraient sur du **logiciel embarqué** (avion, "box" Internet, motorisation automobile, machine à laver, . . .)
- mais **certains logiciels travaillent sur** (ou génèrent, ou transforment, ou analysent . . .) **du logiciel**
- des dizaines de **couches logicielles différentes**

La complexité du logiciel

Indécidabilité du problème de l'arrêt ; démonstrations Turing & Church (1936)

Il n'existe **pas d'algorithme** qui **décide si un logiciel ne boucle jamais** (ou n'est pas "bogué")

Un logiciel est conceptuellement **un texte** (ou un énoncé, voire une démonstration mathématique)

La crise du logiciel (E.Dijkstra 1972) - **complexité du développement**

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful ! To put it quite bluntly : as long as there were no machines, programming was no problem at all ; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, **programming has become an equally gigantic problem**.

Cette **crise est encore actuelle** (et les gros ordinateurs dont parlait E.Dijkstra en 1972 sont mille fois moins puissant qu'un téléphone bas de gamme en 2015)

Code source et code machine

code source C++	**code machine** hexa, x86-64
`extern int counter;`	`8b 05 02 34 00 00`
`while (true) {`	`83 c0 01`
`  counter++;`	`89 05 02 34 00 00`
`}`	`eb ef`

(en réalité, un logiciel est **beaucoup plus complexe**)

Le **code source** est **la forme préférée des développeurs** de logiciel.
Le **code machine** est **la *seule forme* traitable directement par le microprocesseur**.

Ce programme boucle indéfiniment (avec `counter` prenant en quelques secondes toutes les 2^{32} valeurs possibles de 0 à $2^{31} - 1$ puis de -2^{31} à 0 et ainsi de suite ...). Son seul effet de bord (inintéressant, donc ignoré) est de faire chauffer le processeur.

Le code machine a perdu certaines informations ou connaissances **utiles aux développeurs** (par exemple le *nom* -humainement significatif- de la variable `counter`) mais **superflues pour la machine**

modification du code source

Petite modification : Le développeur remplace `true` par `counter < 5` et insère la ligne `cout << ' ' << counter;` pour coder :

```
extern int counter;
while (counter < 5) {
  counter++;
  cout << ' ' << counter;
}
```

Une ***petite* modification** du code source entraine un **changement *drastique* du comportement** de la machine. Elle affiche maintenant : `1 2 3 4` puis **le programme s'arrête** (en moins d'une milliseconde), en s'appuyant sur *plusieurs couches* logicielles [7].

7. Pour les connaisseurs : la `libstdc++`, la `libc`, le noyau Linux, l'émulateur de terminal `gnome-terminal`, le serveur de fenêtres `X11`, le bureau Gnome et son window-manager, les pilotes graphiques, etc....

les outils logiciels : le compilateur & les langages

La transformation du code source en code binaire est complexe (car une transformation naivement réalisée ligne par ligne[8] donnerait un code machine très inefficace) ; il faut utiliser un logiciel complexe appelé **compilateur** (transformant, avec d'autres outils logiciels, le code source en code binaire).

La plupart des langages de programmation[9] ont leur compilateur propre (ainsi le compilateur d'Ocaml n'a rien de commun avec celui pour C++). **Certains languages** (PHP ou Python par exemple) ne sont pas compilés mais **sont interprétés** - ils s'exécutent bien plus lentement (10 à 100 fois plus lents), mais ça peut ne pas être important.

Le bon choix d'un language de programmation[10] (et des couches logicielles à utiliser) est **déterminant**.

---

8. pour ceux qui codent sous Linux, `tinycc` est un compilateur naif pour le langage C. Il est rapide à compiler mais produit du code machine 2 à 10 fois plus lent que `gcc` ...
9. Un language de programmation est une spécification technique abstraite, dans un document en plusieurs centaines de pages en anglais.
10. Il existe plusieurs milliers de langages de programmation différents. Les plus utilisés ne sont pas les plus productifs.

la complexité des logiciels

Il existe des tas (centaines de milliers, voire plus) de logiciels différents (dans plusieurs versions). Il est délicat de les faire travailler ensemble en réseau (le métier d'administrateur système).

Les logiciels sont de plus en plus gros ; quelques ordres de grandeur :

- un projet d'un très bon étudiant de Master 2 ou d'un élève ingénieur en informatique : mille à deux milles lignes de code.
- le **noyau Linux** : douzaine de millions de ligne de code source
- le **navigateur** Firefox : **cinquantaine de millions de ligne de code source**
- le compilateur GCC : dizaine de millions de ligne de code source
- le patrimoine de MicroSoft (secret) estimé à moins d'un milliard de lignes de code - c'est moins que la taille d'une distribution Linux récente (Ubuntu 14, Debian 7, Fedora 22...)

Une automobile récente embarque une centaine de millions de lignes de code (principalement Linux et Firefox).

Le **code critique d'un avion de ligne** : 100 000 à 300 000 lignes de code source (mais **un seul bogue fait crasher l'avion**). Le code non critique : centaines de millions de ligne de code.

le développement d'un logiciel

Il obéit à la *loi de Hofstadter*

Souvent le développeur de code source travaille sur un logiciel existant :

- correctif de bogues (la majorité du coût d'un logiciel)
- ajout de fonctionalités
- test de logiciel
- portage (sur d'autres machines ou systèmes d'exploitation)

Les qualités d'un développeur :

- **sauter** facilement d'un **niveau d'abstraction** à un autre
- écrire du **code lisible** et maintenable, savoir **lire le code d'autrui**
- intuition pour **déboguer** (corriger des bogues $\approx$ en ajouter d'autres)
- **collaborer** avec ses confrères
- suivre l'**évolution technologique**
- **apprendre** des nouveaux **langages de programmations** et des nouvelles couches logicielles ("cadriciel" = "software framework")

la complexité des logiciels selon les exigences

`http://xkcd.com/1425/` sous license CC BY-NC 2.5

Les 4 libertés de l'utilisateur de logiciel

http://www.gnu.org/philosophy/

- la **liberté d'exécuter le programme** comme vous voulez, pour **n'importe quel usage** (liberté 0) ;
- la **liberté d'étudier le fonctionnement du programme**, et **de le modifier** pour qu'il effectue vos tâches informatiques comme vous le souhaitez (liberté 1) ; l'**accès au code source** est une condition nécessaire ;
- la **liberté de redistribuer des copies**, donc d'aider votre voisin (liberté 2)
- la **liberté de distribuer aux autres des copies de vos versions modifiées** (liberté 3) ; en faisant cela, vous donnez à toute la communauté une possibilité de profiter de vos changements ; l'accès au code source est une condition nécessaire.

NB : La FSF prend le **point de vue de l'utilisateur** (averti) d'un logiciel (cet utilisateur pourrait *sous-traiter* les aspects techniques, par exemple s'il ne sait pas développer)

Importance du logiciel libre

Experience de pensée : si le logiciel libre disparait brusquement, sous toutes ses formes (code source et code binaires)

- **Internet et les télécommunications s'arrêtent** (donc aussi les banques)
- les **réseaux d'énergie** (électricité, pétrole) ou d'eau tombent
- la plupart des **ordinateurs ne peuvent plus fonctionner :**
  - les serveurs sur Internet (à 90% sous Linux) et les supercalculateurs (secteur pétrolier, nucléaire, biologie, aéronautique)
  - les téléphones et tablettes ne fonctionnent plus
  - les ordinateurs (même sous Windows, dont TCP/IP est du logiciel libre, ou Apple)
  - systèmes embarqués dans voitures, métros, super-pétroliers, bus, appareils médicaux, . . .
- **on ne peut plus fabriquer des ordinateurs**
- **on n'a** presque **plus de compilateurs**, donc les développeurs ne peuvent plus travailler à refaire le logiciel libre
- **l'humanité recule de plus de cent ans en un instant**

C'est moins vrai pour le logiciel propriétaire ! Pour la plupart d'entr'eux, on peut **immédiatement les remplacer** par du libre (mais *coûts de migration*).

Les logiciels libres qu'on utilise sans le savoir

- l'**infrastructure Internet** : DNS (`bind`), SMTP (courriel : `exim`, `postfix` etc...), HTTP (serveurs web pour plus du 3/4 d'un milliard de sites : `apache`, `nginx`), NTP (heure `ntpd`) - presque toujours du libre
- les **sites Web** : souvent LAMP = Linux + Apache + MySQL + PHP
- les **box internet** -et les routeurs wifi- sont *toutes* sous Linux (ou BSD)
- la **téléphonie** "smartphone" et **tablettes** : Android (Linux), IOS (noyau XNU), FireFoxOS (Linux) ont au moins la couche basse en logiciel libre, hélàs la couche apparente est propriétaire
- la smart **TV** (sous Linux)
- les **liseuses** sont sous Linux (et les **GPS** l'étaient aussi)
- **automobiles** : Linux renommé Genivi
- les aspirateurs robots
- en médecine : **équipements médicaux** à l'hôpital

Certains équipements ont du logiciel libre en couche basse difficile à modifier (dangers de la **TIVOisation**)

Les logiciels libres qu'on devrait utiliser en le sachant

- il est **facile** d'**installer une distribution Linux** sur tout ordinateur PC perso (portable ou fixe). Les distributions *récentes* (LinuxMint, Ubuntu, Fedora, Debian, ...) sont très faciles à installer (depuis 2 ou 3 ans) - parait-il plus faciles qu'installer Windows. **Sauvegardez vos données** préalablement.
- on peut **acheter son ordinateur** sous Linux (ou sans aucun OS) et c'est **moins cher** qu'un PC Windows
- en donnant son ancien ordinateur : installer Linux (et effacer ainsi le disque dur)
- Si on est obligé de garder Windows (par exemple pour certains jeux - mais SteamOS arrive ! ! !) on peut **installer des applications libres** LibreOffice ou OpenOffice comme suite bureautique, Mozilla Firefox comme navigateur, Gimp comme traitement d'images, LaTeX pour rédiger un mémoire ou une thèse scientifique
- ne pas attendre d'un logiciel libre les fonctionalités et l'apparence semblables aux logiciels propriétaires qui vous asservissent ; **regarder le logiciel libre avec des yeux neufs**.

Participer et contribuer au logiciel libre

Si **vous n'êtes pas développeur** :

- **parlez en** (à vos amis, sur vos blogs, réseaux sociaux)
- prenez le temps de **faire des rapports de bogues ou d'anomalie** et suggérez des nouvelles fonctionalités
- **favorisez les formats de données ouverts**
- **participez aux forums** (discussions sur l'utilisation néophyte ou avancée)
- **contribuez à la documentation**, par exemple tutoriels, traductions
- participez au logiciel : **internationalisations des messages d'erreur**, **graphismes**
- **militez** pour le logiciel libre (APRIL, AFUL, FramaSoft, ...)

Si **vous êtes** (même un peu) **développeur** :

- **corrigez les bogues**
- **ajoutez des fonctionalités**
- **participez au développement**

Dans votre vie professionnelle

- **ne violez pas**, ne faites pas violer **les licenses**
- **utilisez et aidez à l'installation du logiciel libre**
- **expliquez le logiciel libre** à votre management et à vos collègues
- si vous êtes développeur : **contribuez des patches** avec l'accord de votre employeur

développement de gros logiciels libres

Pour les gros logiciels libres (noyau Linux, navigateur Firefox, cadriciel Qt, compilateurs GCC ou Clang, suites bureautiques, . . .) de plusieurs millions de lignes :

- la **compréhension du logiciel libre est difficile** (à cause de sa taille)
- les **développeurs sont généralement très qualifiés** et **sont payés à travailler à temps plein** (ou plus qu'à mi-temps) sur un seul logiciel libre.
- les **communautés** de développement sont **très exigeantes et méritocratiques** avec plusieurs centaines de membres actifs
- le premier patch de 10 lignes peut représenter plus d'un mois de travail
- **le logiciel évolue tellement** qu'on doit y consacrer plus de la moitié de son temps ; suivre le travail des autres peut représenter plus d'un tiers-temps !
- souvent il faut **une relation contractuelle formalisée** avec le logiciel et son organisation (FSF, consortium Eclipse, ...)

développement de logiciels libres moyens ou petits

Pour les logiciels libres de quelques centaines de milliers de lignes (ou moins)

- on peut s'y plonger en quelques semaines
- on peut contribuer des patchs en y travaillant par exemple une journée par semaine
- on peut le faire en dehors du travail
- les communautés de développement sont souvent plus petites (une dizaine de développeurs) donc plus agréables

PS. **La métrique "nombre de lignes de code"** n'est pas bonne et **n'est pas pertinente** : un logiciel peut progresser en diminuant de taille ("software refactoring")

Concours de logiciels libres pour étudiants

Ils s'addressent à des étudiants **développeurs**, et peuvent vous financer :

- **Google Summer Of Code** `https://www.google-melange.com/` (une "fédération" de projets) - "bourse" de 5000$US
- **INRIA Boost Your Code** (contrat d'un an)

Attention, les deux sont des concours sélectifs (s'y préparer 6 mois à un an à l'avance). Il vous faudrait avoir *déjà* fait un peu de logiciel libre (dans une petite communauté).

Plus tard, dans votre vie professionnelle (pour les entreprises) : appels européens H2020, Systematic FUI, ...

catalogues de logiciels libres

Il y en a beaucoup.

En français : ceux de l'APRIL, AFUL, FramaSoft, Systematic GdT logiciels libres

En anglais : sites **`http://sourceforge.net/`**, **`http://github.com/`** etc etc etc...

Ceux des distributions Linux.

Le logiciel libre : **coûteux à développer**, quasi-**gratuit à distribuer**
(le logiciel propriétaire est encore plus coûteux à développer et se distribue difficilement)

Plusieurs **dizaines de milliers** de **logiciels libres**

Variété des logiciels (libres ou propriétaires)

Différences qualitatives et quantitatives :

- applications smartphone (**“time to market”**), **bogue sans importance**
- logiciels applicatifs “bureautique” : **plantage parfois acceptables**
- logiciels embarqués non-critiques (“box”) : **plantage tolérable**
- logiciels sur serveurs : **plantage embêtants ou intolérables**
- logiciels embarqués critiques (avion) : **le bogue tue** ;
  certification/validation exigée par le législateur

Différentes méthodologies de développement :

- développement agile
- développement communautaire
- développement sous procédure formalisée (DO-178C en avionique)
  (on rédige un rapport de 10 pages pour modifier 5 lignes de code ; on refait des tests, y compris sur banc d’essai et des vols d’essai)
- **méthodes formelles** (on prouve le logiciel avec l’aide d’un analyseur de code) : parfois utiles, mais pas la panacée (la preuve d’un gros logiciel de millions de lignes est hors de portée)

Internet et les logiciels libres

Dépendance(s) mutuelle(s) circulaire(s) entre les deux :

- **Internet n'existerait pas** (ou plus) **sans logiciel libre**.
- **le logiciel libre n'existerait pas sans Internet** :
  - importance des **communications par Internet** ("mailing lists", "forums", IRC, ...) **pour les communautés de développeurs**
  - **diffusion par Internet** des logiciels libres
  - **outils logiciels libres** pour le support et le développement des logiciels libres :
    - compilateurs et logiciels associés (environnement de développement)
    - système d'exploitation GNU/Linux
    - **partage du code source en cours de développement** dans des dépôts de versionneur comme `git`
    - gestionnaires de bogues et **forges logicielles**
  - **recherche** [11] et développement nécessaires : outillage libre et collaboratif (par exemple : language de programmation facilitant le développement communautaire et outillage collaboratifs associés)

11. La recherche actuelle est à la science ce que le fast-food est à la nourriture. Sortir du dogme "publish or perish" et de la recherche effrénée du projet pour son financement ; avoir des projets scientifiques et/ou technologiques sur 7 à 10 ans....

Il faut encore plus de logiciels libres

- pas (ou peu) de **logiciels libres embarqués critiques** (avion de ligne, ferroviaire, médical, ...)
- le vote électronique (mais l'urne transparente est souvent préférable)
- **variété insuffisante des logiciels libres d'infrastructure** (système d'exploitation, couches basses, Internet) - **risque systémique** ? ?
- **manque d'offre logiciels libres pour certaines applications**
- **les outils logiciels devraient être repensés et reconçus**
  (il n'existe aucun compilateur parallèle, donc le développeur est comme le cordonnier mal chaussé)
- **avénement du parallèlisme**
  (il faudrait repenser et refaire tous nos logiciels pour le multi-cœur et le Cloud)
- pour l'économie : **gain de productivité** par **l'externalité positive de réseau**.
- le logiciel libre favorise **l'emprunt et le partage de code**
- plus de **projets logiciels libres dans la durée** (i.e. 4 à 7 ans : on ne réalise pas le même logiciel avec 2 ans + 2 ans qu'avec 4 ans) - concevoir et faire *ex nihilo* un OS libre comme GNU/Linux (qui a 25 ans ! !) ? ?