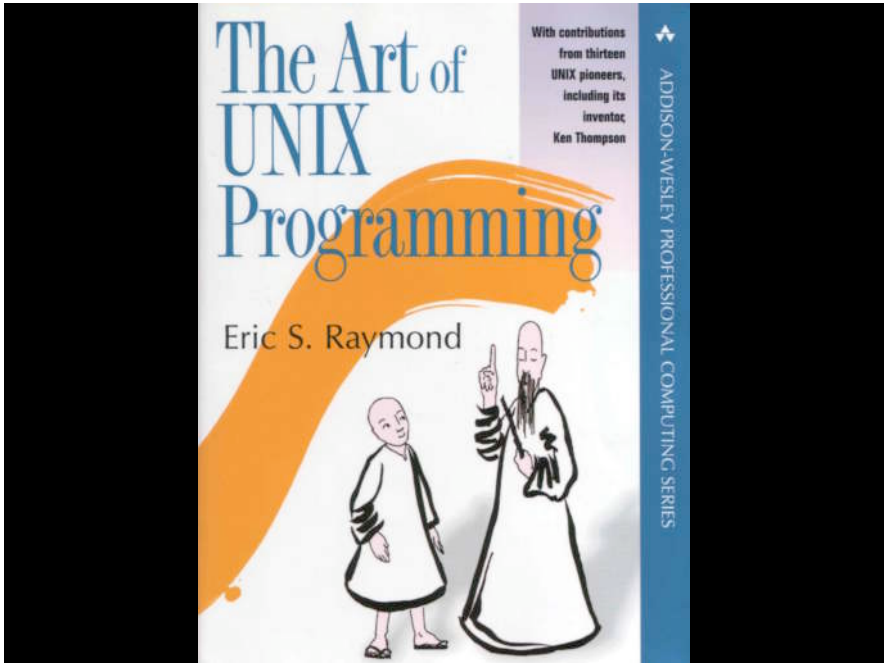
L'art de programmer: l'expérience Unix

par

Marc Parizeau



35 années de culture Unix

- Logiciel libre («open source»)
- Portabilité multi-plateforme
- Standards ouverts
- Internet & WWW
- Flexibilité

Les trois générations

✓Premier prototype

---> **CTSS** = Compatible Time-Sharing System

✓Deuxième version: *obésité* caractéristique

---> projet **Multics**

✓Troisième version: recherche de la *simplicité*

---> **Unix**

Histoire

pdp-7

- ✓1969: naissance d'Unix
  ---> Ken Thompson (Bell Lab.)
- ✓69-71: la genèse
- ✓71-80: l'exode
- ✓80-90: TCP/IP et la grande guerre
- ✓90-95: le mouvement «open source»
  ---> naissance de Linux (1991)

Langage C
(1973)

Ken & Dennis devant un pdp-11 (1972)

- ✓Inventé par Dennis Ritchie
- ✓Basé sur le langage B
- ✓A permis de ré-écrire Unix pour pouvoir le porter facilement sur plusieurs plateformes
- ✓Standard ANSI en 1986

AT&T vs BSD

- 1958: AT&T interdit de vendre des ordinateurs
- 1977: première version Unix BSD (Berkeley)
- 1981: l'équipe BSD choisie pour TCP/IP
- 1982: SUN = «*dream machine*»!
- 1983: AT&T est charcuté en 15 morceaux
  ---> levée de l'interdit
  ---> licence Unix = 40K$!
- 1985: projet GNU (R. Stallman, MIT)
  ---> «*Free Software Foundation*»

Linux

- Introduit en 1991 en version beta;
- par Linus Torvalds
  ---> étudiant à la maîtrise, université finlandaise
  ---> dictateur bienveillant!

Philosophie Unix

- «*Write programs that do one thing and do it well*»;
- «*Write programs that work together*»;
- «*Write programs to handle text streams, because that is a universal interface*».

Doug McIlroy

---> On peut en déduire *17 règles fondamentales*

1: Modularité

- Écrire de *petits modules* branchés à travers des *interfaces simples*
- Limiter la *complexité* est l'essence de la programmation!

2: Clarté

- La *clarté* est préférable à l'*ingéniosité*.
- Écrire les programmes non pas pour la machine, mais pour l'*humain* qui doit *maintenir le code*.
- Du code *élégant* et *clair* comporte un *risque plus faible* de contenir des bogues.

3: Composition

- Concevoir les programmes pour qu'ils puissent se *combiner* avec d'*autres programmes*.
- *Lire* et *écrire* du *texte ascii* dans un *flot* d'entrée/sortie.
- Éviter les *hypothèses* sur l'identité de celui qui écrit en entrée et de celui qui lit en sortie.

4: Séparation

- Séparer les *politiques* des *mécanismes*.
- Les *mécanismes* de base ont habituellement une *durée de vie* beaucoup *plus longue* que les *politiques*!

5: Simplicité

- Concevoir pour la *simplicité*; ajouter de la complexité seulement si on ne peut pas faire autrement!
- Éviter les *modes* qui poussent vers toujours *plus* de *fonctionnalités*.

6: Parcimonie

- Écrire de *gros programmes* seulement lorsqu'il est démontré qu'on ne peut pas procéder autrement! (rarement)

7: Transparence

- Concevez votre code pour qu'il soit aisément *lisible* et *compréhensibles*.
- Un logiciel est «*transparent*» lorsqu'on peut le comprendre par simple inspection du code.
- On dit qu'il est «*observable*» lorsqu'il incorpore des *mécanismes* qui permettent de *monitorer* son *état interne*.

8: Robustesse

- La *robustesse* résulte de la *transparence* et de la *simplicité*.
- Un logiciel robuste *performe* correctement même en présence de *conditions inhabituelles* ou *inattendues*.
- Éviter autant que possible les *cas spéciaux* dans votre code.

9: Représentation

- Représenter la *connaissance* dans les *données* au lieu des *programmes*.
- Chercher à déplacer la *complexité* du code vers les *données*!

10: Moindre surprise

- En concevant les *interfaces*, choisir l'alternative la plus *naturelle*, la plus *intuitive*.

11: Silence

- Un programme qui n'a rien à *signaler* devrait se *taire*.

12: Réparation

- Un programme devrait *réparer* les *erreurs*! Mais s'il doit échouer, alors il devrait le faire *tôt* et *bruyamment*.
- Soyez *libéral* en entrée et *conservateur* en sortie!

13: Économie

- Le temps du *programmeur* coûte plus *cher* que celui de la *machine*.

14: Génération

- Écrire des programmes qui *génère* du *code*!
- Permet d'élever le *niveau d'abstraction*.

15: Optimisation

- Faire fonctionner le *prototype* avant d'*optimiser*.
- «Premature optimization is the root of all evil» (D. Knuth, grand maître des algorithmes)
- «Make it run, then make it right, then make it fast» (Kent Beck, grand maître de la «*programmation extrême*»)

16: Diversité

- Se méfier des *solutions* dites *universelles*.
- Engendre souvent des logiciels *obèses* et *rigides,* incapables de communiquer avec le reste du monde.

17: Extensibilité

- Concevoir pour le *futur* car il sera là bien plus tôt que vous ne le pensez!

Conclusion

✓Lire le livre: http://www.faqs.org/docs/artu/

✓«*programmation extrême*»

✓«*modélisation agile*»