
Table des matières

Avant-propos	xvii
<i>par Greg Wilson</i>	
Préface	xix
1. Un moteur d'expressions régulières	1
<i>par Brian Kernighan</i>	
Pratique de la programmation	2
Implémentation	3
Discussion	4
Autres possibilités	6
S'en servir comme fondation	7
Conclusion	9
2. Le delta-éditeur de Subversion : ontologie de l'interface	11
<i>par Karl Fogel</i>	
Contrôle de versions et transformation d'arbres	12
Exprimer les différences entre deux arbres	16
Interface du delta-éditeur	17
Est-ce bien de l'art ?	24

	L'abstraction vue comme un spectacle sportif	26
	Conclusions	29
3.	<i>Le plus beau code que je n'ai jamais écrit</i>	31
	<i>par Jon Bentley</i>	
	Le plus beau code que j'aie jamais écrit	31
	De plus en plus, avec de moins en moins	33
	Mise en perspective	39
	En quoi consiste exactement l'« écriture » ?	41
	Conclusion	42
	Remerciements	43
4.	<i>Chercher et trouver</i>	45
	<i>par Tim Bray</i>	
	À temps	45
	Problème : les données d'un weblog	46
	Problème : qui a consulté quoi, et quand ?	55
	La recherche en production	60
	Conclusion	62
5.	<i>Correct, beau et rapide (dans l'ordre) : leçons enseignées par la conception de vérificateurs XML</i>	63
	<i>par Elliotte Rusty Harold</i>	
	Rôle de la validation XML	63
	Le problème	64
	Première version : implémentation naïve	66
	Deuxième version : imiter la grammaire BNF. Complexité : $O(N)$	67
	Troisième version : première optimisation. Complexité : $O(\log N)$	69
	Quatrième version : deuxième optimisation. Pas besoin de vérifier deux fois.	70
	Cinquième version : troisième optimisation. Complexité : $O(1)$	73
	Sixième version : quatrième optimisation : utilisation du cache	78
	La morale de cette histoire	80
6.	<i>Framework pour tests intégrés : la beauté dans la fragilité</i>	81
	<i>par Michael Feathers</i>	
	Un framework de tests de validation composé de trois classes	82
	Les défis posés par la conception d'un framework	84
	Un framework ouvert	85
	Un parseur HTML réduit à sa plus simple expression	87
	Conclusion	89

7. <i>Les beaux tests</i>	91
<i>par Alberto Savoia</i>	
Cette irritante recherche dichotomique	92
Présentation de JUnit	95
Faire un sort à la recherche dichotomique	97
Conclusion	110
8. <i>Génération de code à la volée pour le traitement d'images</i>	111
<i>par Charles Petzold</i>	
9. <i>Précédence des opérateurs du haut vers le bas</i>	135
<i>par Douglas Crockford</i>	
JavaScript	136
Table des symboles	137
Jetons	138
Précédence	139
Expressions	140
Opérateurs infixes	140
Opérateurs préfixes	143
Opérateurs d'affectation	143
Constantes	144
Portée	144
Instructions	146
Fonctions	150
Littéraux de tableaux et d'objets	151
Pistes de réflexion et d'améliorations	152
10. <i>La quête d'une distance de Hamming accélérée</i>	153
<i>par Henry S. Warren, Jr.</i>	
Techniques élémentaires	154
Diviser pour régner	155
Autres méthodes	157
Somme et différence des poids de Hamming de deux mots	159
Comparer le poids de Hamming de deux mots	159
Compter les bits activés dans un tableau	160
Applications	165
11. <i>Communications sécurisées : la technologie de la liberté</i> ..	169
<i>par Ashish Gulhati</i>	
Principes fondateurs	170
Démêler l'écheveau de la messagerie sécurisée	172
La clé, c'est l'ergonomie	173

Les fondations	176
La suite de tests	180
Le prototype opérationnel	181
Nettoyage, branchement, démarrage... ..	182
Intervenir depuis l'Himalaya	186
La main invisible	191
La vitesse, ça compte	193
Secret des communications et libertés individuelles	194
Hacker la civilisation	195
12. <i>Faire pousser du beau code en BioPerl</i>	197
<i>par Lincoln Stein</i>	
BioPerl et le module Bio::Graphics	198
Le processus de conception de Bio::Graphics	202
Plus loin que Bio::Graphics	221
Conclusions et leçons apprises	225
13. <i>Conception du trieur de gènes Gene Sorter</i>	227
<i>par Jim Kent</i>	
Interface utilisateur de Gene Sorter	228
Maintenir un dialogue avec l'utilisateur sur le Web	229
Un peu de polymorphisme peut emmener loin	231
Filtrer pour ne conserver que les gènes pertinents	234
Théorie du beau code en production	235
Conclusion	238
14. <i>Comment un code élégant peut évoluer avec le matériel : le cas de l'élimination Gaussienne</i>	239
<i>par Jack Dongarra et Piotr Luszczek</i>	
L'impact de l'architecture des ordinateurs sur les algorithmes matriciels	240
Une approche basée sur la décomposition	242
Une version simplifiée	243
Le sous-programme DGEFA de LINPACK	245
Le sous-programme DGETRF de LAPACK	248
Décomposition LU récursive	251
Le sous-programme PDGETRF de ScaLAPACK	254
Le multithreading pour les systèmes multicœurs	259
Un mot sur l'analyse des erreurs et le comptage des opérations	262
Orientations futures pour la recherche	263
Références bibliographiques	264

15. <i>Les bénéfiques à long terme d'une conception esthétique</i>	265
<i>par Adam Kolawa</i>	
Ma conception de la beauté dans un code	265
La bibliothèque du CERN	266
La beauté extérieure d'un code	267
La beauté intérieure d'un code	274
Conclusion	280
16. <i>Le modèle de pilote du noyau Linux ou les bénéfiques d'un travail collectif</i>	281
<i>par Greg Kroah-Hartman</i>	
Des premiers pas modestes	282
Comment faire plus court ?	287
Des milliers de périphériques à gérer	290
Des petits objets faiblement couplés	292
17. <i>Un autre niveau d'indirection</i>	293
<i>par Diomidis Spinellis</i>	
Du code aux pointeurs	294
Des arguments de fonctions aux pointeurs d'arguments	295
Des systèmes de fichiers aux couches de systèmes de fichiers	300
Du code à un langage propre au domaine	301
Multiplexer et démultiplexer	304
Jamais les couches ne se couchent ?	305
18. <i>Implémentation des dictionnaires Python : être tout pour tous</i>	307
<i>par Andrew Kuchling</i>	
À l'intérieur du dictionnaire	309
Provisions spéciales	311
Collisions	312
Changement de taille	313
Itérations et modifications dynamiques	314
Conclusion	315
Remerciements	315
19. <i>Itérateurs multidimensionnels en NumPy</i>	317
<i>par Travis E. Oliphant</i>	
Les défis fondamentaux dans la manipulation des tableaux à N dimensions	318
Modèles mémoire pour un tableau à N dimensions	319
Naissance de l'itérateur de NumPy	320

Conception de l'itérateur	321
Interface de l'itérateur	327
Utilisation de l'itérateur	328
Conclusion	332
20. Un système d'information d'entreprise hautement fiable pour la mission Mars Exploration Rover de la NASA	333
<i>par Ronald Mak</i>	
La mission et le portail d'information collaboratif (CIP)	334
Les besoins de la mission	335
Architecture du système	337
Une étude de cas : un service de flux	340
Fiabilité	342
Robustesse	350
Conclusion	353
21. ERP5 : concevoir pour une adaptabilité maximale	355
<i>par Rogerio Atem de Carvalho et Rafael Monnerat</i>	
Objectifs généraux d'un ERP	356
ERP5	356
La plate-forme Zope sous-jacente	358
Concepts du projet ERP5	362
Programmation d'ERP5 Project	363
Conclusion	368
22. Une cuillerée d'eaux usées	369
<i>par Bryan Cantrill</i>	
23. Programmation distribuée avec MapReduce	387
<i>par Jeffrey Dean et Sanjay Ghemawat</i>	
Un exemple illustratif	387
Le modèle de programmation MapReduce	390
D'autres emplois du modèle MapReduce	392
Une implémentation MapReduce distribuée	393
Extensions du modèle	397
Conclusion	398
Pour en savoir plus	398
Remerciements	398
Annexe : programme de comptage de mots	399
24. Une merveilleuse concurrence	401
<i>par Simon Peyton Jones</i>	
Un exemple simple : les comptes bancaires	402
L'approche STM (Software Transactional Memory)	405

Le problème de Santa Claus (problème du Père Noël)	414
Réflexions sur le langage Haskell	422
Conclusion	423
Remerciements	425
25. <i>Abstraction syntaxique : le processeur syntax-case</i>	427
<i>par R. Kent Dybvig</i>	
Introduction rapide à syntax-case	431
Algorithme d'expansion	434
Exemple	446
Conclusion	449
26. <i>Une architecture économe : un framework orienté objet pour applications réseau</i>	451
<i>par William R. Otte et Douglas C. Schmidt</i>	
Exemple d'application : un service de journalisation	453
Conception orientée objet du framework du serveur de journalisation	455
Implémenter des serveurs de journalisation séquentiels	462
Implémenter des serveurs de journalisation concurrents	467
Conclusion	472
27. <i>Intégration des partenaires métier avec REST</i>	475
<i>par Andrew Patzer</i>	
Contexte du projet	476
Fourniture de services à des clients externes	476
Routage du service au moyen du design pattern Fabrique	480
Échange de données au moyen des protocole e-business	482
Conclusion	487
28. <i>Beau débogage</i>	489
<i>par Andreas Zeller</i>	
Déboguer un débogueur	490
Un processus systématique	492
Un problème de recherche	493
Trouver automatiquement la source de la panne	494
Débogage par deltas	496
Minimiser les données en entrée	498
À la chasse au défaut	499
Un problème de prototype	502
Conclusion	502
Remerciements	503
Pour aller plus loin	503

29. <i>Traiter le code comme une dissertation</i>	505
<i>par Yukihiro Matsumoto</i>	
30. <i>Quand votre seul lien au monde se résume à un bouton ...</i>	511
<i>par Arun Mehta</i>	
Modèle de conception élémentaire	512
Interface de saisie	515
Efficacité de l'interface utilisateur	528
Téléchargement	528
Perspectives d'évolution	528
31. <i>Emacspeak : le bureau auditif complet</i>	531
<i>par T. V. Raman</i>	
Produire une sortie vocale	532
Doter Emacs de la parole	533
Accéder sans peine aux informations en ligne	544
Résumé	551
Remerciements	554
32. <i>Code en mouvement</i>	555
<i>par Laura Wingerd et Christopher Seiwald</i>	
Pourquoi organiser le code comme un livre ?	556
Qui se ressemble... se ressemble	558
Les dangers de l'indentation	559
Naviguer dans le code	560
Les outils employés	561
Le passé mouvementé de DiffMerge	563
Conclusion	564
Remerciements	565
Pour aller plus loin	565
33. <i>Écrire des programmes pour le « Livre »</i>	567
<i>par Brian Hayes</i>	
La voie roturière	568
Avis à ceux qui prennent les parenthèses en grippe	568
Trois d'un coup	569
Une pente glissante	571
L'inégalité triangulaire	573
Les méandres de ma réflexion	575
« Bon sang ! » — ou plutôt « Eurêka ! »	576
Conclusion	577
Pour aller plus loin	578

Postface	581
<i>par Andy Oram</i>	
Contributeurs	583
Index	593

