**L’HYPERPUISSANCE DE L’INFORMATIQUE**

*Document rédigé par Ignace Rak en décembre 2018 pour l’association PAGESTEC* [*www.pagestec.org*](http://www.pagestec.org)*,*

Mots-clés sur <http://pagesperso-orange.fr/techno-hadf/index.html> : informatique

Pour éviter les répétitions du nom de Gérard Berry, j’emploie pour simplifier, l’abrégé G.B.

Les titres et sous-titres du résumé ci-dessous, sont ceux figurant dans son ouvrage (1).

Cette troisième partie de résumé de l’ouvrage de G.B, fait suite à celle rédigée en novembre 2018 (2).

***Avertissement***

Comme les résumés des ouvrages précédents (3) concernant le futur prévisible sur les contenus d’enseignement rédigés pour les professeurs de technologie de l’association PAGESTEC (4), ce sont des résumés de cet ouvrage que je vous propose = du chapitre 9 à 10 de cet ouvrage de plus de 500 pages.

Les extraits de l’ouvrage sont indiqués entre parenthèses et en écriture en italique. Des relectures attentives peuvent cependant avoir laisser échapper des erreurs de saisie. Seul l’ouvrage original fait foi.

**3ème PARTIE**

**Rendre l’informatique plus sûre**

Gérard Berry décrit des problèmes rencontrés en informatique et leurs solutions. Il traite deux aspects importants concernant d’une part, des bugs et trous de sécurité qui sont deux dangers de l’informatique, et d’autre part comment rendre l’informatique plus sûre.

**Chapitre 9** (5)

**Bugs et trous de sécurité : deux dangers de l’informatique**

*« Passons maintenant à deux cotés sombres mais incontournables de l’informatique, ceux liés aux bugs et aux trous de sécurité ses systèmes informatisés…Les deux aspects sont souvent reliés, car les trous de sécurité sont le plus souvent liés à des microbugs passés inaperçus car n’étant pas reliés au fonctionnement normal et donc pas détectés par les tests fonctionnels classiques, souvent restreints aux cas considérés comme normaux…Tous les bugs n’ont bien sûr pas la même importance…*

*Le niveau le plus exigeant est celui des systèmes* safety critical, *ceux qui mettent directement en jeu des vies humaines. Ces systèmes se trouvent par exemple en avionique, ferroviaire et nucléaire…*

*Les processus de certification peuvent être internationaux, comme en avionique, ou nationaux, comme en nucléaire (ce qui m’étonnera toujours) …*

*Le deuxième niveau concerne les systèmes appelés* mission critical*, ceux pour qui la présence de bugs peut conduire à l’abandon de missions couteuses mais sans impact direct sur les vies humaines. Un bon exemple est celui de systèmes spatiaux. Les méthodes de développement sont généralement proches de celles des systèmes critiques, mais sans processus de certification externe.*

*Le troisième niveau est celui des systèmes* business critical*, ceux pour qui les bugs peuvent tuer un produit ou même une compagnie. On ajoute parfois le qualificatif* time critical*, pour dire que le projet doit impérativement se terminer avant une certaine date…*

*Un point qui m’étonne toujours est que l’automobile semble encore échapper à la classification ci-dessus, et en particulier à ses deux niveaux supérieurs, alors même que les voitures sont de plus en plus informatisées, y compris pour leurs fonctions critiques…Mais la principale raison me semble être que l’industrie automobile est encore guidée par un schéma mental qui est celui de la mécanique. Si les problèmes liés à la mécanique se détectent avec nos yeux ou nos oreilles, ce n’est absolument pas le cas pour ceux dus à l’informatique, où les inspections visuelles ou auditives sont inopérantes. Les problèmes liés aux logiciels sont d’autant plus dangereux que les bugs sont reproduits à l’identique dans toutes les voitures d’un type donné. J’espère que nos autorités se saisiront un jour de ces questions surtout avec l’arrivée de systèmes d’aide à la conduite de plus en plus perfectionnés, visant même des formes d’autonomie plus ou moins complètes de la voiture… ».*

**Les bugs de temps ne sont pas rares**

***Le Zune et la PS3 Fat n’aimaient pas les années bissextiles***

*« Notre premier exemple sera celui du Zune, lecteur MP3 commercialisé en 2006 par Microsoft. Le 31 décembre 2008 à minuit, tous ces appareils se sont arrêtés brutalement de fonctionner, n’obéissant à aucune commande. En fait, ils n’étaient pas arrêtés, mais simplement en train de vider leur pile sans rien faire d’utile. Recharger la pile et redémarrer son Zune le 1er janvier faisait immédiatement réapparaitre le même problème. Il fallait attendre le lendemain pour que l’appareil se remette en marche normale, du moins jusqu’au 31 décembre 2012 à minuit où le même phénomène ne manquerait pas de se reproduire de façon déterministe.*

*Que s’était-il passé ? Un bug idiot dans un programme tout simple écrit en langage C par un contractant, qui était en charge de l’heure. Pour convaincre définitivement le lecteur qu’un seul caractère suffit à changer un bon programme en un gros bug, je choisis de lui montrer le petit bout de code concerné, qui ne demande qu’une compréhension minimale de la programmation… ».*

Pour celles et ceux qui sont intéressés pour intégrer cela dans leurs cours au collège, ou pour en construire d’autres sur ce sujet, G.B. montre les morceaux de programmes concernés (les variables en cause, la programmation et la boucle « tant que » concernées), on les invite à lire la suite. Et pour d’autres exemples sur les bugs de temps, voir ce que cite G.B. :

***Le bug du réveil de l’Iphone 4***

***Un bug du mercredi***

***Les Raptor n’aiment pas la ligne de changement de date***

***Le Patriot rate le scud***

**Le bug qui a tué Ariane 501**

De la même façon que précédemment, G.B. entre un peu dans la technique. Nous avons relevé quelques points clés. Et nous conseillons aux professeurs intéressés, de lire directement les autres détails techniques dans l’ouvrage.

« *Peu de systèmes complexes sont fabriqués avec autant de soins que les systèmes spatiaux, tous jugés* mission critical. *Pourtant les bugs informatiques n’y sont pas rares…Le plus célèbre est celui de l’échec du premier lancement Ariane 501…*

*Le début du vol fut nominal, mais après 37 secondes, la fusée braqua brutalement et commença à se disloquer…*

*Il s’agissait d’un bug minuscule dans un logiciel embarqué dans les composants essentiels que sont les* systèmes de référence inertielle (SRI)., *capteurs à base de gyrolasers qui déterminent l’attitude (NDLR altitude ?) et la vitesse du lanceur, donc des paramètres essentiels pour son guidage ».*

***Les SRI se déclarent en panne par erreur***

*Comme beaucoup de composants clefs des systèmes spatiaux, ces SRI sont doublés, avec un SRI actif et un SRI de secours identique qui calcule en même temps mais dont les données ne sont utilisées qu’en cas de panne détectée sur le premier…*

*Après 36,5 secondes de vol, le SRI de secours a détecté une erreur interne et s’est déclaré en panne. Le SRI actif a fait la même chose immédiatement après. Dès qu’il a enregistré la panne du SRI actif, le calculateur de bord a basculé sur le SRI de secours, qui lui a renvoyé un message d’erreur au lieu d’un angle au lieu d’un angle puisqu’il était aussi en panne… ».*

G.B. explique dans le détail (qu’il faut lire pour celles et ceux qui sont intéressés pour utiliser l’exemple dans leurs cours) qu’ « *il n’y avait en fait aucun problème car les deux SRI fonctionnaient parfaitement…La cause de l’arrêt des deux SRI était identique, et liée à une variable flottante. À exactement 36,7 secondes de vol, cette variable avait dépassé 65 535, qui est le plus grand nombre entier stockable sur 16 bits…Son stockage en mémoire flash a alors provoqué ce qu’on appelle un débordement arithmétique …Le débordement arithmétique a provoqué ce que l’on appelle une* exception logicielle. *Si n’est pas traitée explicitement, comme c’était le cas pour les SRI, une telle exception arrête immédiatement l’exécution du programme ADA tout entier. Les deux SRI se sont donc arrêtés quasi simultanément, puis déclarés en panne. Mais en fait ils ne l’étaient en fait pas,* car la valeur stockée ne servait à rien, et tout ce qui servait fonctionnait normalement ! *Le vrai bug était d’avoir détecté un faux bug…*

***Les vraies causes du problème étaient humaines***

*Le vrai problème n’était en fait pas dans l’erreur de conversion mais dans les causes humaines. Comme je l’ai déjà dit et répété, le bug n’est pas une panne de la machine, mais la conséquence d’une erreur humaine de conception - en fait ici d’une série d’erreurs provenant d’une même cause, la sous-estimation des difficultés du logiciel et de la transposition au logiciel d’un schéma mental pertinent pour la matière mais inadapté à l’immatériel…*

***Analyses et recommandations de la commission d’enquête***

*La commission d’enquête a émis de nombreuses recommandations pour la correction des logiciels d’Ariane 5, et, de façon plus générale, pour le développement et la validation des logiciels spatiaux…arrêter toute fonction logicielle non indispensable pendant le vol, comme ici le mode de calibration des SRI, d’interdire aux équipements de cesser d’envoyer les meilleures données disponibles, ce qui aurait suffit à garder le vol 501, de confiner les exceptions à l’intérieur des tâches et de prévoir des procédures de secours en cas d’exception logicielle…*

*En mathématiques, quand un auteur annonce un théorème, des relecteurs vérifient systématiquement les démonstrations avec le plus grand soin, car on estime qu’on ne peut pas faire confiance à une seule personne pour affirmer qu’elles sont justes, surtout si c’est l’auteur ; il faut faire pareil pour la programmation des systèmes spatiaux….*

***Un impact positif : la vérification par interprétation abstraite***

*L’explosion d’Ariane 5 a aussi eu un impact heureux sur le passage des techniques modernes de vérification de programmes de la recherche à l’industrie. En 1977, Patrick et Radhia Cousot avaient publié un article maintenant célèbre proposant une nouvelle méthode mathématique de vérification appelée interprétation abstraite…Une implémentation prototype de l’interprétation abstraite par le regretté Alain Deutsch de l’équipe Cousot a permis de trouver automatiquement et rapidement le bug des SRI, rien qu’en analysant leur code…*

*A l’aide d’un nouveau système appelé Astrée, l’équipe de Patrick Cousot à l’École normale supérieure a pu garantir à 100% l’absence d’erreurs à l’exécution dans l’intégralité du programme de pilotage de l’Airbus A380, gros code de centaines de milliers de lignes…*

***Faut-il dupliquer les logiciels pour se protéger des pannes ?***

*Un système embarqué doit bien sûr se protéger des pannes toujours possibles. Un choix classique pour un lanceur de courte durée de vie est de doubler les équipement matériels critiques, avec un actif et un de secours comme pour les SRI d’Ariane. Si l’actif est en panne, on passe sur le secours. Si celui-ci est aussi en panne, on considère que l’on ne peut plus rien faire et la mission est abandonnée. Quand on considère les seules pannes matérielles, c’est la bonne méthode si ces pannes sont indépendantes. Dans ce cas, on peut multiplier les probabilités des pannes individuelles pour calculer la probabilité de panne...*

*Mais ce raisonnement ne marche plus du tout pour un bug logiciel comme celui des SRI d’Ariane 5. À moins qu’elles n’aient explosé ou trop dévié de leur trajectoire auparavant,* toutes *les fusées auraient explosé après 37 secondes de vol. Le logiciel est déterministe…*

*On peut envisager deux méthodes pour sortir du problème. La première est de* garantir formellement que le logiciel n’a pas de bug *(d’un certain type) …La seconde est de* diversifier les logiciels. *C’est la solution prise par Airbus pour ses avions depuis l’A320 (1987) … ».*

***Les bugs ont-ils une probabilité ?***

Dans ce paragraphe G.B. à la question posée par des non informaticiens lors de l’expertise de logiciels critiques *« …Quelle est la probabilité qu’il y ait un bug dans ce logiciel ? ... », et à cette affirmation « … tout a une probabilité … », « …Pour le matériel, oui. Pour la logique, l’informatique, pas forcément… ».*

**Quelques bugs martiens**

*« Les missions sur Mars sont particulièrement coûteuses et traitées avec beaucoup de soin. Mais les bugs logiciels n’y ont pas été rares, voire plus nombreux que les problèmes matériels. Certains ont pu être réparés de façon acrobatique grâce à la compétence des ingénieurs, d’autres non… ».*

De la même façon que pour Ariane 5, G.B. analyse les différents bugs sur les vaisseaux spatiaux cités ci-dessous. Nous vous invitons à lire la description un peu plus technique de ces incidents dans les paragraphes ci-dessous.

***L’inversion de priorité de Paphfinder***

***La mémoire flash trop pleine qui a failli tuer Spirit***

***La disparition de Mars Climate Orbiter***

***Le crash de Schiaparelli***

**Avionique et ferroviaire, des domaines critiques et sérieux**

*Les avions ont été informatisés depuis longtemps. Pour les avions de chasse, rendus instables dans 1950 afin d’augmenter leur maniabilité, c’était indispensable car aucun humain ne pouvait les piloter. Pour les avions de ligne, l’informatisation massive a été initiée par l’Airbus A320, dont le premier vol date de 1987…*

*L’informatique est devenue ainsi un des points clefs de la sécurité. Par ailleurs, il ne faut pas oublier que les avions sont désormais conçus de façon entièrement numérique, comme nous l’avons vu au chapitre 2…*

*L’informatisation a joué un rôle prépondérant dans la diminution des accidents d’avions, hors bien sûr actions humaines délétères…*

*Lors du fameux accident d’un Airbus A320 mont Sainte-Odile en 1992, un des facteurs identifiés était une possibilité de confusion dans la lecture des écrans informatisés du cockpit. Un écran affichait un paramètre de descente qui pouvait être soit en degrés soit en mètres par seconde, mais la distinction n’était pas affichée de façon suffisamment claire sur l’écran et le pilote a rentré une consigne dans le mauvais mode. Il faut bien comprendre que l’ergonomie des cockpits d’avion est un sujet difficile, où l’informatique ne résout pas tout. C’est la même chose pour les tours de contrôle…*

**L’automobile, un domaine hélas moins sérieux**

*Pour les voitures, les choses se sont passées bien différemment. L’informatique embarquée a été introduite par les équipementiers et non par les constructeurs eux-mêmes…ABS, …régulateur d’allure, …injection électronique, …airbags, …radar ou caméra arrière, …climatisation, …tableau de bord évolué, …autoradio numérique perfectionné, etc…*

*L’automobile évolue maintenant vers une assistance à la conduite de plus en plus perfectionnée avec le projet à court ou à moyen terme de fabriquer des voitures sans conducteur, donc vraiment autonomes… ».*

G.B. développe quelques exemples qu’il faut aller lire dans les paragraphes ci-dessous.

***Quelques bugs automobiles***

***Pourquoi le moteur ne s’arrête-t-il pas ?***

***Portes boquées***

***Le contrôle moteur meurtrier des Toyota Camry***

***Pourquoi l’automobile n’a-t-elle pas encore le sérieux nécessaire ?***

« …*l’automobile diffère profondément de l’avionique et du ferroviaire par trois caractéristiques : premièrement c’est une industrie très concurrentielle à grands volumes et faibles marges, donc essentiellement dirigée par les coûts ; ensuite comme elle reste mentalement centrée sur la mécanique, il est encore souvent jugé que développer des logiciels est beaucoup plus facile, qu’innover en mécanique, et ne pose pas de problème particulier ; enfin, les problèmes et accidents de voitures sont dilués et ne font jamais la une des journaux, contrairement aux accidents d’avions ou de trains – d’ailleurs, la compréhension de tout ce qui concerne l’informatisation des objets reste bien faible dans les mondes du journalisme et de la communication*…

*Enfin dans les accidents d’avions ou de trains, on désigne immédiatement des commissions d’enquête compétentes avec lesquelles les constructeurs sont tenus de collaborer. Ce n’est pratiquement jamais le cas pour l’automobile…* ».

**Un bug médical historique : le Therac-25**

*« …Je vais parler ici brièvement d’un cas historique, celui de la machine de radiothérapie Therac-25 fabriquée par AECL (Atomic Energy of Canada Limited) qui a provoqué ou accéléré la mort d’au moins 5 patients et gravement blessé plusieurs autres de juin 1985 à janvier 1987…*

*Le Therac-25 pouvait fonctionner selon deux modes exclusifs, délivrant des faisceaux de rayons X ou d’électrons très précis en géométrie et en dosage…Le therac-25 était une amélioration d’une machine précédente, le Therac-20, dans laquelle le bon positionnement de la table tournante était garanti par des dispositifs mécaniques. Pour le Therac-25, le fabricant avait décidé d’informatiser le système avec une interface par clavier et écran informatique standard commandant numériquement la machine…*

*J’en viens maintenant aux accidents, tous en lien avec l’interface homme-machine et la sécurité gérée par logiciel. L’interface avait visiblement pas mal de défauts, dont celui de donner très souvent des erreurs du type « MALFUNCTION XX » où XX était un numéro d’erreur, sans que l’opérateur ne dispose de la moindre documentation sur ces erreurs. Dans ce cas, il pouvait appuyer sur la touche « ↑ » du clavier pour relancer l’opération. Au bout de cinq essais sans succès, il fallait redémarrer la machine …*

*Un premier accident se produisit le 3 juin 1985 aux Etats-Unis, brûlant profondément une patiente avec des séquelles graves et irréversibles…Un autre accident eu lieu en Ontario en juillet 1985. Quand le traitement fut démarré, la machine s’arrêta au bout de cinq secondes avec un message « NO DOSE », indiquant qu’aucune dose n’avait été délivrée. L’opérateur appuya sur « ↑ », reçut le même message, et recommença la même chose quatre fois avant que la machine ne s’arrête en demandant son redémarrage. Mais le patient se plaignait de fortes douleurs et les médecins comprirent vite qu’il avait été sévèrement surirradié les cinq fois…*

*Quand on appuie sur une touche, le traitement logiciel à effectuer n’est pas instantané et peut dérouler un nombre important d’instructions machine. Il est essentiel que ce traitement soit* atomique*, c’est-à-dire que l’ensemble de ces instructions soit exécuté avant de commencer quoi que ce soit d’autre…Ce problème, connu depuis toujours, se gère parfaitement à l’aide de sémaphores d’exclusion mutuelle ou* mutex*, comme nous l’avons vu pour Pathfinder et Spirit. La façon dont ceux-ci étaient implémentés dans le système d’exploitation dédié du Therac-25 était incorrecte…*

**Les bugs d’outillage**

***Les compilateurs C sont-ils fiables ?***

*…Toute personne qui écrit un programme en C, ce qui est très fréquent pour le logiciel embarqué, s’attend à ce que ce programme en langage machine obtenu par la compilation de son programme fasse exactement ce qu’il a programmé. Or, un bug dans le compilateur peut conduire à un code machine faux sans que l’utilisateur ne puisse s’en apercevoir…*

*Un projet de recherche américain très intéressant a montré les limites des hommes dans l’écriture des compilateurs, alors même que ce domaine fait partie des plus sérieux de l’informatique (yang, Chen, Eide et Reger, 2011) …*

**Quid *des systèmes d’exploitation ?***

*Les systèmes d’exploitation ont un rôle central dans la gestion des machines…Un bug particulièrement remarquable a eu lieu sur le bateau de guerre USS Yorktown…informatisation massive…pour…une flexibilité maximale…*

*Lors de sa première sortie suivant son informatisation un opérateur a entré par mégarde le nombre 0 dans champ un champ de formulaire. Cela provoqué une division par 0, erreur normalement anodine si elle est bien gérée. Cela n’a pas été le cas : tous les systèmes informatiques du bord se sont progressivement arrêtés, provoquant même l’arrêt de la propulsion, et il n’a pas été possible de les remettre en fonction ! Le bateau a dû être tracté jusqu’au port…*

*Windows NT et son cousin Windows CE dédié à l’embarqué n’étaient certes pas réputés très solides à l’époque…Linux reste un modèle de sureté.*

**Bugs des systèmes distribués**

***Le crash du téléphone interurbain d’AT&T***

*Le 15 janvier 1990, à 2 h 25 de l’après-midi, le téléphone interurbain d’AT&T aux Etats-Unis s’est effondré, 60 000 personnes n’ayant plus accès au réseau de 70 millions d’appels n’ayant pu aboutir pendant la période de la panne. La moitié des 144 centraux passaient leur temps à redémarrer.*

*Il a fallu plusieurs semaines à AT&T pour détecter et corriger le problème. Que s’était-il passé ? Juste un bug aussi minuscule que stupide, sous la forme d’une instruction pensée inutile mais en fait fausse ajoutée lors d’une modification des logiciels des centraux, installée en novembre de l’année précédente pour rendre le traitement des appels plus rapide…*

*Pour les connaisseurs du langage C : le bug était dû au remplacement par le « programmateur inconnu » d’une branche* else *absente d’un test* if *par « else break ; ». Le programmateur croyait que le break terminerait le* if, *ce qui n’est pas le cas en C.* … ».

Nous laissons le lecteur intéressé lire la suite de ce développement technique de G.B.

***L’autodéni de service de Facebook ( ?)***

« *J’ai le souvenir d’un jour où, à cause d’un bug probablement du même tonneau, Facebook s’est trouvé en autodéni de service pendant quelques heures…*

**Bugs de circuits**

*Les circuits sont conçus et fabriqués avec un très grand soin, beaucoup plus grand que la plupart des logiciels. En effet, au contraire des logiciels, il n’est pas possible de les corriger une fois qu’ils ont été fabriqués et installés dans les machines…*

***Le fameux bug de division du Pentium Pro***

*Le Pentium Pro était le nouveau microprocesseur phare d’Intel. Quelque temps après sa sortie, Thomas Nicely, mathématicien américain spécialiste de théorie des nombres, mit en évidence un bug étonnant. La division flottante 1/824 633 702 441 rendait un résultat faux…Le bug est bien trop subtil pour être décrit ici, mais il est passionnant pour qui aime les nombres…Intel reconnut évidemment le bug…mais minimisa son importance…*

*Mais cet argument ne marcha pas du tout…Intel se trouva obligé de proposer le changement de les pentium Pro livrés. Au total, le bug lui couta officiellement 475 millions de dollars…*

*C’est d’ailleurs ce bug qui a vraiment lancé le domaine de la preuve formelle de circuits arithmétiques, comme quoi les mauvais bugs ont aussi du bon ! ...*

**Des bugs aux trous de sécurité**

*Longtemps traitée par-dessus la jambe, la sécurité informatique est maintenant prise au sérieux, grâce (si l’on peut dire) aux exploits des hackers, toujours plus forts pour pénétrer les systèmes informatisés…*

***Les attaques par débordement de mémoire***

*Hormis celles des virus qui savent exploiter des bugs profonds dans les systèmes d’exploitation mais sont trop compliqués pour être décrits ici, les premières attaques on souvent été liées à des bugs des navigateurs Web, programmes souvent développés à « l’arrache » pour gagner rapidement des parts de marché. Par exemple, Internet Explorer de Microsoft a souffert de très nombreux débordements de buffers, analogue au bug du mercredi présenté au début de ce chapitre…*

***Les attaques par injection légale de codes malveillants***

*Les navigateurs Web, gestionnaires de courrier ou utilitaires comme les visualisateurs de documents PDF, constituent d’autres points d’entrée potentiels pour les hackers, car ils peuvent exécuter des programmes inclus dans les pages pou documents qu’ils traitent. En cas de bugs de ces outils, ces programmes peuvent exécuter des actions malveillantes sans que l’utilisateur s’en aperçoive…*

*Les bugs qui permettent d’entrer profondément dans les systèmes ou les outils qu’ils hébergent sont souvent assez simples mais pas détectables par les tests fonctionnels habituels…*

***Le danger des systèmes d’exploitation obsolètes***

*Beaucoup de systèmes informatisés encore en activité ou même tout récents sont vulnérables parce qu’ils fonctionnent au-dessus de systèmes d’exploitation périmés, par exemple Windows XP dont Microsoft a abandonné la maintenance en 2014 en déconseillant formellement toute utilisation ultérieure. En mai 2017, l’attaque WannaCry s’est répandue à très grande allure en chiffrant le contenu d’ordinateurs tournant sur Windows XP ou sur des versions postérieures par gardées à jour par les utilisateurs. De nombreux hôpitaux ont été touchés en Angleterre, et de nombreuses infrastructures industrielles de pays. On a parlé de 300 000 « victimes » …*

***Les faiblesses des protocoles de sécurité***

*Regardons maintenant les trous de sécurité dus aux connexions réseaux. Le fondement des communications sécurisées de type https pour le Web est le* chiffrement *des données avant leur transmission et de leur* déchiffrement, *ces deux calculs étant réalisés par des opérations arithmétiques sur les messages numérisés reposant sur des choix de clefs numériques de chiffrement et de déchiffrement…les vraies faiblesses…sont le plus souvent dans les bugs des protocoles de sécurité qui préparent l’établissement des communications sécurisées.*

*Ces protocoles ont pour but l’établissement de canaux de communication cryptées entre leurs utilisateurs. Le plus connu est TLS (transport layer security) : vous le mettez en jeu dès que vous essayez de vous connecter à un site Web sécurisé dont l’adresse commence par « https// » …*

*TLS est un protocole complexe dont certains restent encore fragiles. Il fait donc l’objet d’un grand nombre d’attaques détectées ou grâce aux travaux de chercheurs…*

*La première est Heartbeed, due à un bug introduit dans sa version publique très utilisée OpenSSL,…La faille touchait à peu près de 10% des téléphones et tablettes fonctionnant sous Android…*

*La deuxième est Freak …créée par des chercheurs de Microsoft Research, de l’INRIA et de plusieurs universités américaines. Cette attaque profite des règles de maintien de compatibilité édictées par la NSA (National Security Agency) américaine pour rester compatible avec de vieux protocoles…Par une attaque de type* man in the middle*, il est possible de forcer une transaction à opérer avec un de ces vieux protocoles, puis de casser sa clé courte…*

*La troisième attaque, du même type, est Logjan, due à une équipe de l’INRIA Nancy, qui se fonde cette fois sur une faille du protocole lui-même et pas seulement de certaines implémentations, en amenant TLS à utiliser un vieux chiffrement utilisant seulement des nombres premiers de 155 chiffres décimaux, ce qui est insuffisant…*

***Les attaques par des objets***

*Les dénis de service distribués (*DDoS = distributed denial of service*)* *font partie des attaques les plus violentes auxquelles sont confrontés les sites et services Web, voire des pays tout entiers. Elles consistent à noyer ces sites ou services sous un nombre gigantesque de requêtes envoyées simultanément par un très grand nombre d’ordinateurs, qu’il n’est même pas nécessaire de posséder pour les attaquants…*

*En 2016 a eu lieu une attaque de très grande envergure sur le site américain Dyn, qui est un très important site DNS (*domaim name server*) traduisant les noms de site, comme* [*www.collège-de-france.fr*](http://www.collège-de-france.fr) *en adresse IP numériques comprises par le réseau. Les objets utilisés ont été les caméras vidéo de surveillance, installées à marche forcée ces dernières années ; leurs fabricants n’avaient pas pensé à cet usage détourné…car ajouter des couches de sécurité aux logiciels embarqués est difficile, coûte cher, …*

***Les attaques sur les objets***

*…J’ai parlé précédemment des bugs des voitures. Leurs problèmes de sécurité ne sont pas moins nombreux... »*

G.B. en développe quelques-uns de ces incidents « … *l’ouverture facile et à distance des portes ; mises à jour via le Web et l’autoradio des logiciels embarqués qui permettent de prendre le pouvoir à distance sur les fonctions essentielles de ces voitures : mettre les essuie-glaces et le lave glace en route, … mettre l’auto radio à la puissance maximum, … freiner à fond ou désarmer les freins, …mettre le moteur à fond ou le couper, et plus récemment, prendre le contrôle de la direction. Cela a conduit Chrysler à rappeler 1,5 millions de voitures…*

*La domotique est aussi un domaine où les objets sont porteurs de promesses, mais aussi de dangers. Un bon exemple est donné par le système SmartThings de Samsung destiné à écouter et commander tous les objets connectés d’une maison…des chercheurs de l’Université du Michigan ont mis en évidence de sérieux points faibles dans le système…Ils ont ainsi montré qu’ils pouvaient déclencher les alarmes à volonté, mettre le four à fond, et surtout injecter leurs propres codes dans les serrures pour entrer et sortir quand bon leur semble…*

*Dans le domaine médical, les attaques peuvent être très dangereuses. Une bien jolie concerne les pacemakers : en 2008, des chercheurs de la Johns Hopkins University ont montré qu’il n’était pas difficile de prendre le pouvoir total sur les pacemakers d’un firme leader du domaine à l’aide d’une simple télécommande…*

***Méfiez-vous des messages d’erreurs !***

*Plusieurs sites Web et applications de téléphone ont été évidents à percer à cause d’un message d’erreur émis trop précocement lors de la frappe du mot de passe. Ils trouvaient malin de signaler l’erreur au premier caractère erroné du mot frappé. Pour simplifier, supposons que le mot de passe ait 8 lettres majuscules ou minuscules. Il y a dans ce cas 52 possibilités par position, donc 528 différents, de l’ordre de 50 000 milliards. Pas facile à casser, sauf bien sûr si votre mot de passe est dans le dictionnaire. Mais si le message st émis dès le premier caractère faux, trouver le mot de passe est trivial : il suffit d’essayer au plus 52 caractères pour la première position jusqu’à ne plus recevoir de message d’erreur, puis les 52 suivants une fois connu le premier caractère accepté, etc…*

*Le deuxième est plus subtil. C’est celui du nouveau passeport français, pour lequel un bug de sécurité a été prouvé par les chercheurs du groupe de sécurité CNRS/INRIA… »*

**Chapitre 10** (6)

**Comment rendre l’informatique plus sûre ?**

*« Après avoir présenté cette liste d’échecs ou de problèmes dus à des bugs identifiés comme tels après coup, abordons des questions centrales de l’informatique : comment concevoir et mettre en place des systèmes informatisés qui marchent comme nous le souhaitons, sans être affectés, voire détruits par des bugs que y avons-nous-mêmes introduits ?...*

*À l’heure actuelle, on leur applique trois types de réponses : d’abord mieux travailler pour réduire le nombre de bugs dès la conception ; ensuite, vérifier en permanence son travail pour attraper les bugs avant la mise en service, soit par des batteries de tests, soit à l’aide de méthodes formelles présentées dans ce chapitre ; troisièmement, mettre des bugs résiduels suffisamment tôt avant que leurs conséquences ne deviennent catastrophiques, ce qui permet aussi de transmettre de bons rapports de bugs aux concepteurs…*

**Un préliminaire indispensable : caractériser les bons et mauvais comportements**

***Cas 1 : on peut spécifier complètement l’application***

*Ce premier cas est celui on l’on sait dire exactement ce qu’il faut faire et ne pas faire, autrement dit définir précisément les critères de succès et d’échec. Le problème y est bien posé et la notion de bug bien carrée. Parmi les bugs du chapitre précédent…ceux liés au temps, celui de l’arithmétique du Pentium Pro, celui du téléphone longue distance d’AT&T, ou ceux du compilateur C …*

***Cas 2 : on peut spécifier des propriétés partielles à respecter absolument***

*Ce second cas, très fréquent mais moins carré, est celui des programmes pour lesquels on ne sait pas dire dans tous les détails ce qu’il faut faire ou ne pas faire, mais où on sait en revanche donner des listes de contraintes sur le comportement à respecter absolument. En d’autres termes, seuls certains critères de succès ou d’échec peuvent être définis proprement.*

*On divise habituellement les contraintes à satisfaire en deux types : contraintes de* sureté*, qui expriment que certains mauvais comportements ne peuvent pas exister, e, contrainte de* vivacité*, qui expriment que certains bons comportements doivent absolument finir par se produire. Prenons l’exemple d’un ascenseur. Deux contraintes typiques de sûreté sont que les cabines ne doivent jamais voyager la porte ouverte et qu’elles ne peuvent pas continuer à monter après le dernier étage. Deux contraintes de vivacité sont qu’au moins une cabine doit atteindre tout étage où l’ascenseur a été appelé et y ouvrir ses portes (partie de la contrainte qu’il est facile d’oublier pour un débutant) …*

*Deux difficultés supplémentaires apparaissent par rapport au cas 1. D’abord, les contraintes peuvent être nombreuses, et chacune d’entre elles peut être ardue à bien spécifier…Par exemple, pour l’ascenseur, il est essentiel de bien expliciter la liste et les propriétés des capteurs et actionneurs avec lesquels le logiciel dialogue, mais aussi de traduire des lois physiques variées. Un bon exemple pour l’ascenseur allant de l’étage 2 à l’étage 4 doit nécessairement passer par l’étage 3, propriété ancrée dans nos esprits qu’il est fréquent de l’oublier. Or cette propriété apparemment triviale est essentielle pour réduire le nombre de possibilités à explorer par les méthodes de vérification…*

***Cas 3 : la sécurité informatique et la résistance aux attaques***

*…La sécurité étant un sujet particulièrement difficile, les failles sont nombreuses dans les systèmes actuels. Les détecter est l’objet d’une course permanente entre trois populations : celle des agresseurs (hackers) qui cherchent à les exploiter directement, celle des chercheurs qui cherchent à les trouver et à les corriger avant que les agresseurs ne puissent les exploiter, et celle plus récente de ceux qui cherchent à les trouver en premier pour les vendre aux enchères, soit aux constructeurs soit aux agresseurs… ».*

*G.B. pointe aussi les* canaux cachés « *…c’est-à-dire des moyens d’attaque qui n’ont pas été prévus dans la conception du système ou qu’on ne sait pas encore éviter même si on les connait. Un bon exemple de canal caché est fourni par une clef USB qu’on ramasse par terre puis qu’on insère dans un port d’ordinateur…*

*Le même type de difficulté se produit pour un vote « électronique », c’est-à-dire bien entendu géré par un logiciel. J’ai entendu des élus nationaux me dire que c’était un sujet facile car il suffit de compter les oui et les non. J’ai immédiatement répondu que cette vue était totalement simpliste : qu’est-ce qui me prouve que mon vote a bien été pris en compte ? Que personne n’a pu le consulter ? … Que personne n’a pu entrer de faux votes dans le système par un canal caché ? …le vote classique bourrer les urnes…les experts sont clairs : on ne sait pas encore quelle est la bonne liste de critères pour quel type de vote…Or pas mal de bugs et de possibilités de canaux cachés ont été trouvés par les chercheurs…par exemple lors de l’élection entre George Bush et Al Gore en 2000…*

***Cas 4 : ni les comportements ni leurs contraintes ne sont bien définis a priori***

*C’est le cas des systèmes qu’ont ne peut pas vraiment concevoir et mettre en place sans une expérimentation intensive. Un bon exemple est* l’interface homme-machine *(IHM), composant essentiel mais hélas souvent négligé de la plupart des applications. Il est difficile de spécifier les bons comportements à réaliser et les erreurs à éviter sans expérimentation fine et intensive avec des utilisateurs finaux.*

*L’interface homme-machine est en fait intermédiaire entre la science et l’art. beaucoup d’innovations sont dues à de longues expérimentations hors des sentiers battus. Par exemple, le poste de travail moderne avec sa puissance de calcul, son écran graphique, sa souris, ses disques locaux et sa connexion réseau avec les autres postes….*

*Il est important de constater que les interfaces homme-machine ne concernent pas que les ordinateurs et téléphones : les cockpits d’avion, les centres de commande d’usine, les postes de travail des aiguilleurs du ciel et les interfaces pour la chirurgie robotisée…jamais faciles à expliciter. Souvenez-vous que l’accident de l’Airbus A320 qui s’est écrasé sur le mont Sainte-Odile a été attribué en bonne partie à une présentation et une entrée ambiguë des données de vol sur les écrans du cockpit…*

***Cas 5 : les algorithmes d’apprentissage***

*…les voitures autonomes guidées par des réseaux profonds analysant les données produites par les caméras ou des lidars semblent bien se comporter sur la route. Mais, en 2017, les systèmes d’apprentissage ne sont pas assez répandus pour qu’on puisse en dire beaucoup plus pour l’instant…*

*Presque rein de ce que j’ai raconté sur les bugs dans les quatre cas précédents ne semble valable ici…Comment va-t-on donner le permis de conduire à une voiture autonome…Comment vont se comporter les conducteurs face aux voitures autonomes, que ce soit la leur ou celles des autres ? Vont-ils être capable de reprendre la main en cas d’imprévu, ou n’auront-ils plus la vigilance nécessaire, et sauront-ils quoi faire s’ils ne savent pas du tout comment les automatismes fonctionnent ? De même, comment va se comporter un médecin face à un diagnostic automatisé, s’il n’est pas sûr d’être d’accord avec lui ? Comment l’expliquera-t-il au patient, et qui sera responsable des mauvais diagnostics ?...*

**Le génie logiciel**

*L’énumération de bugs et trous de sécurité ci-dessus du chapitre précédent donne bien entendu une image partiale de l’informatique, insistant trop sur ses mauvais côtés. Heureusement, il est tout à fait possible de construire des applications qui marchent bien, en particulier dans les deux premiers cas de l’énumération précédente. Pour cela, il est indispensable de maitriser le génie logiciel, ensemble de méthodes et procédures qui permettent de conduire efficacement le développement et la vérification des applications, …*

***Mieux spécifier***

*Le premier et peut-être le plus difficile des problèmes se place au niveau des spécifications. On les décrit souvent comme une réponse à la question : « Que doit faire le logiciel ? », en pensant que quelques documents textuels ou graphiques suffisent à y répondre. C’est très insuffisant, pour plusieurs raisons. D’abord, l’expérience montre que les documents de ce type contiennent fréquemment des* incomplétudes *et des* incohérences*…*

*Il ne faut jamais oublier que les spécifications ne doivent pas se contenter de décrire la tâche à effectuer. Elles doivent aussi décrire précisément dans quel environnement cette tâche doit s’exécuter et définir des contraintes non fonctionnelles comme les limites en temps de calcul, taille mémoire et énergie consommée, surtout lorsque ces contraintes sont critiques pour la correction pratique de l’implémentation.*

*Par exemple, pour les applications temps réels liées au monde physique, « avant l’heure, c’est l’heure, après l’heure c’est plus l’heure » : quand on appuie sur la pédale de frein d’une voiture, l’effet doit être quasi instantané, et quand on veut que deux notes de musique apparaissent simultanées, leurs ordres de déclenchement ne doivent pas être séparées de plus d’une quinzaine de millisecondes. Les spécifications doivent aussi définir ce qui doit se passer lorsque les commandes envoyées au système ne sont pas conformes à ce qui est attendu, mais quand même possibles. Beaucoup de trous de sécurité majeurs résultent d’un manque d’analyse de cet aspect…*

*Enfin, une des causes récurrentes de l’échec de développements informatiques est le changement permanent des spécifications, très fréquent quand les donneurs d’ordre n’ont pas eux-mêmes une connaissance suffisante de la difficulté et de la lourdeur de l’informatique : changer les spécifications est un acte léger pour eux, mais pas pour les implémenteurs chez qui mettre à jour l’implémentation est les tests est toujours un acte lourd…*

*…une question toujours délicate est la suivante : les spécifieurs et les implémenteurs doivent-ils être des personnes différentes ? Cela dépend des applications. Par exemple, pour le pilotage d’un avion, cette distinction est la plupart du temps naturelle : les spécifieurs sont des automaticiens, spécialistes su contrôle, et les implémenteurs des informaticiens…C’est la même chose pour les circuits électroniques entre les architectes et les designers (programmeurs en langage de description de circuits).*

*En revanche dans des domaines où la programmation et l’expérimentation sont intimement reliées comme l’IHM (interface-homme-machine), ce sont souvent les mêmes individus qui gèrent les deux aspects. Jean-Marie Hullot, principal créateur des interfaces de ma machine NeXT puis du système Mac OS X du Macintosh et de l’iPhone d’Apple, est à la fois un artiste des interfaces et un programmeur hors norme.*

***La gestion informatique de spécifications, programmes et tests***

*Pour tout projet de quelque envergure, gérer proprement et donc en machine les spécifications, programmes est absolument indispensable. Ça l’est en particulier pour garder une mémoire précise des développements permettant de comprendre pourquoi telle ou telle décision a té prise, afin de pouvoir revenir en arrière en cas de mauvaise décision…*

***Assurer la traçabilité de la spécification à l’implémentation***

*L’exigence de traçabilité des décisions de la conception à la réalisation est une contrainte essentielle de toutes les procédures de certification de logiciels…Elle n’a vraiment de sens que si de bons documents de spécification existent…*

***Conduire des revues externes***

*Les concepteurs et réalisateurs d’un projet ne sont pas les mieux placés pour analyser la façon dont ils travaillent et les résultats qu’ils produisent. Il est souvent utile d’organiser des revues et vérifications indépendantes par des consultants ou organismes extérieurs. C’est d’ailleurs obligatoire dans les domaines certifiés comme l’avionique, le spatial, le ferroviaire ou le nucléaire – mais hélas pas pour l’automobile. L’échec d’Ariane 501 et les accidents provoqués par le Therac-25 et le contrôle moteur de Toyota sont significatifs de ce point de vue…*

**Du test aux méthodes formelles**

*Le test est la méthode la plus utilisée pour mettre au point les applications informatiques…*

***Le test par simulation su système global***

*…Par exemple, en avionique on construit des simulateurs très perfectionnés qui permettent de « faire voler numériquement » l’avion en partant de la conception mécanique 3D et de modèles mathématiques de son comportement aérodynamique…Les maquettes physiques disparaissant petit à petit…Un avantage est de pouvoir simuler un grand nombre de pannes ou de situations difficiles, ce qui est souvent impossible en vol réel…*

***Tester ou prouver ?***

*Mais le test garde une insuffisance fondamentale qu’il ne faut jamais ignorer, et qui a bien été mise en lumière par la fameuse citation d’Edsger Dijksra (prix Turing 1962) : « Le test de programmes est une manière très efficace de montrer la présence de bugs, mais il est désespérément inefficace pour montrer leur absence. La seule façon vraiment efficace d’augmenter significativement la confiance dans un programme est de donner une preuve convaincante de sa correction » …*

**Les assertions logiques**

*Pour l’informatique, le papier fondateur de l’idée de preuve de programme est encore une fois dû à Alan Turing…*

*Turing établissait (ainsi) un lien fort et applicable en pratique entre la programmation des ordinateurs et les mathématiques. Clarifier, perfectionner et automatiser ce lien allait devenir le sujet de beaucoup de recherches, dont certaines des miennes…Les assertions constituent maintenant une façon fondamentale d’intégrer la vérification à la programmation, pouvant comlémenter le test et même parfois le remplacer totalement…Pour expliquer les assertions, il faut un minimum de technique – du niveau lycée rassurez-vous. La figure 70 montre un programme de calcul itératif simple de la factorielle d’entier positif* n*, notée en mathématiques* « n ! »*… ».*

On invite le lecteur intéressé à consulter ce paragraphe, ceci avec l’aide de cette figure et des explicitations détaillées, G.B. sur les assertions logiques des *prédicats* (valeur *vrai* ou *faux*) et des *préconditions* et *postconditions*, permettant de préciser le rôle des assertions logiques.

De la même façon, on invite le lecteur à prendre techniquement la connaissance des contenus des paragraphes suivants,

***Les assertions comme spécifications***

***L’automatisation du calcul des assertions***

***Assertions temporelles et vérification de modèles…***

**Les systèmes de démonstration automatique**

*Le développement de systèmes de démonstration automatique ou assistée de théorèmes mathématiques ou d’assertions de programme a été très tôt un objectif de chercheurs en intelligence artificielle, avant de devenir une discipline spécifique de l’informatique classique…*

***Le problème SAT (satisfaction booléenne)***

*Le* calcul booléen *est le tout bête calcul logique où les variables peuvent prendre les valeurs* vraiou faux*, ou de façon équivalente 0 et 1, et où les opérateurs sont les conjonctions* et*, la disjonction* ou*, et la négation* non*. On y cherche souvent à savoir si la formule donnée est* satisfiable*, c’est-à-dire pouvant être rendue vraie en choisissant bien les valeurs des variables, ou encore* tautologique*, c’est-à-dire vraie pour toutes les valeurs des variables…*

*Le calcul booléen a toujours joué un rôle essentiel dans l’industrie des circuits électroniques, binaires par nature. Mais ce qui le rend fondamental pour la vérification automatique, c’est qu’il permet aussi de coder beaucoup d’autres problèmes…*

*L’histoire du calcul booléen est fascinante. Elle a bien sûr commencé avec George Boole au XIXe siècle, qui a défini ce calcul et fait ainsi passer la logique de la philosophie à l’algèbre et aux mathématiques…Le lecteur intéressé pourra consulter la bibliographie sur les tables de Karnaugh, nées en 1953, qui ont longtemps servi à optimiser une formule booléenne à partir de sa table de vérité*

*La première révolution date de 1971, avec l’introduction par Stephen Cook (prix Turing 1982) de la notion d’algorithme NP-complet qui sera détaillée dans l’annexe dans l’annexe 12.1. Cook a montré que le problème SAT de la satisfaction booléenne était équivalent à un grand nombre d’autres problèmes si on savait le faire pour SAT…Les logiciels SAT modernes arrivent à résoudre des problèmes industriels ayant* des millions *de variables, ce qui était totalement impensable il y a seulement quinze ans. Le mot « industriel » est ici important. On distingue en effet plusieurs classes de problèmes : les problèmes* intrinsèquement durs*, pour lesquels aucun système connu ne passe à l’échelle ; les problèmes* aléatoires*, peu utiles en pratique mais qui intéressent beaucoup pour d’autres raisons les chercheurs en physique statistique ; et les problèmes* industriels*, directement issus de cas pratiques. Pour une raison parfaitement inconnue, ces derniers s’avèrent souvent solubles…*

***La satisfaction modulo théories***

*…Les logiciels de satisfaction modulo théories (De Moura & Bjorner, 2011), en abrégé SMT, permettent de combiner le calcul booléen pour le raisonnement logique et des solveurs spécifiques dédiés à différentes théories décidables utiles en pratique : l’arithmétique de Presburger (celle des nombres entiers limitée à l’addition, la soustraction et la comparaison), la théorie des nombres réels (plus simple, ce qui n’est vraiment pas intuitif), l’arithmétique d’accès aux tableaux, la manipulation de pointeurs en mémoire, etc…*

***Prouver automatiquement la terminaison***

*Prouver la terminaison d’un programme peut être très difficile, voire impossible, comme l’a montré Alan Turing dès 1936. Mais c’est une question très importante en pratique. Prenons par exemple des fameux « écrans bleus » associés aux crashs de Windows, qui ont si longtemps fait perdre des données et du travail à bien des gens. De façon naturelle les utilisateurs les attribuaient à des bugs de Microsoft. Mais c’était loin d’être toujours le cas. Très souvent, le problème venait des* drivers *(« pilotes de périphériques » en français), qui soit crashaient eux-mêmes, soit ne terminaient jamais, provoquant indirectement un crash du système par terminaison d’une temporisation de sécurité contrôlant le driver : il vaut mieux arrêter tout, plutôt que laisser attendre indéfiniment l’utilisateur…*

**Les assistants de preuve**

*Les assistants interactifs de preuve permettent de vérifier des problèmes arbitraires non traitables actuellement par des méthodes automatiques. Comme les solveurs SMT vus précédemment, l’idée de base est de réduire les problèmes de vérification à des problèmes de logique formelle…*

***Les logiques utilisées***

*Comment s’y prend-on pour traiter un programme avec un assistant de preuve ? Il faut d’abord formuler le programme et les conditions à vérifier avec une logique appropriée, ce qui n’est pas toujours simple. Les assistants de preuve sont fondés sur des logiques assez variées, toutes issues des recherches en logique mathématique conduites depuis les années 1930. Les plus simples sont les logiques du* premier ordre*…Les plus riches sont les logiques d’*ordre supérieur…

***La preuve en pratique***

*Une fois la logique et l’assistant choisis, il faut conduire les preuves à l’aide de l’assistant, qui va gérer l’ensemble des formules et des preuves en vérifiant à chaque étape la correction de chaque déduction. Les déductions ne sont heureusement pas faites une par une à la main. Elles l’étaient dans les tout premiers systèmes…*

***La vérification du métro Méteor***

*Le plus ancien logiciel de grande ampleur vérifié formellement à l’aide d’un assistant de preuve a certainement été le logiciel de conduite du métro Meteor de Paris (ligne 14) par la RATP) en utilisant la méthode B de Jean-Raymond Abrial (Abrial 1996). L’histoire est longue et intéressante.
Au début des années 1980, la RATP a décidé de rendre la conduite du RER A de Paris à l’aide d’un système informatique embarqué. Mais sa direction, inquiète de la sécurité du système, commanda à Abrial un audit technique pour s’assurer que la réalisation correspondait bien aux spécifications. Abrial a rapidement répondu qu’il ne pouvait pas répondre à la question, car on ne lui avait jamais montré de spécifications au cours de toutes ses discussions avec les partenaires industriels du projet. A la fureur des industriels qui avaient réalisé le système, mais à juste raison, la RATP décida de mettre le système en service que partiellement : au lieu de transmettre les ordres au matériel, le logiciel les écrivait sur un écran et le conducteur devait les exécuter en vérifiant qu’ils n’étaient pas aberrants*

*Un peu plus tard, la RATP décidé de coopérer avec Abrial pour réaliser le logiciel embraqué de Meteor, qui devait lui être totalement automatique, l’idée était de construire et vérifier le logiciel en utilisant la méthode B d’Abrial, dont la logique est une forme restreinte de théorie des ensembles du premier ordre … »*

G.B. décrit ensuite succinctement le processus de réalisation d’Abrial, description que nous vous engageons à lire si vous êtes intéressés.

« *…En confiance, la RATP prit la décision de supprimer tous les tests unitaires et d’intégration habituels. Le succès fut total, le logiciel n’ayant jamais montré le moindre bug depuis sa mise en service en 1998. La conclusion est nette : pour ce projet, la preuve a coûté beaucoup moins cher que le test. Plus tard la ligne 6 a été automatisée de la même façon…*

***La vérification des circuits***

*J’ai mentionné à plusieurs reprises le coût exorbitant des bugs de circuits. Souvenez-vous par exemple du fameux bug de division du Pentium Pro en 1991 décrit au chapitre 9… ».*

G.B. cite trois exemples où des preuves assistées se sont imposées*.*

***La vérification de systèmes d’exploitation***

*Les systèmes d’exploitation, ainsi nommés, OS pour* opérating systems*, gérent les activités des utilisateurs et les ressources des machines…*

*Mais ces grands systèmes sont mal armés pour faire fonctionner les processeurs des objets embarqués, qui disposent de relativement peu de puissance de calcul. Il existe pour eux une autre race de systèmes qu’on appelle des* micronoyaux*, ou* microkernels *en anglais. C’est pour l’instant sur eux que se sont portés les efforts de preuve…*

***La vérification des compilateurs***

*Au chapitre précédent, j’ai présenté le générateur aléatoire dirigé Csmith de programme C qui a mis en évidence des centaines de bugs dans les compilateurs C principaux, qu’ils soient académiques ou industriels. J’ai aussi mentionné qu’un seul compilateur avait survécu, le compilateur CompCert de Xavier Leroy (aussi auteur de Cmal et OCaml). Ce compilateur a effectivement été complétement prouvé correct en utilisant Coq…*

***Le compilateur C formellement vérifié CompCert***

*Le compilateur C formellement prouvé CompCert a été développé à l’Iniria par Xavier Leroy et son équipe. Prouver un compilateur traitant le langage C complet n’est pas réellement possible car, même s’il est normalisé, C n’est pas un langage bien défini…*

*L’expérience CompCert a été marquante dans le domaine, et d’autres chercheurs cherchent maintenant à prouver des compilateurs pour des langages différents…*

**Les preuves de sécurité**

*En termes de sécurité, tout n’est pas formalisable puisque les canaux cachés sont par nature inconnus. Il faut donc se restreindre à la protection des attaques particulières, par exemple celles consistant à intercepter ou forger des messages lors de la négociation par un protocole de sécurité de type SSL/TLS, ou encore à des contraintes qu’on souhaite respectées pour un système de vote…*

**Quand utiliser la vérification formelle ?**

*La réponse est simple : le plus tôt et le plus longtemps possible, c’est-à-dire dès la mise en forme exécutable des spécifications et tout le long de la programmation et de la vérification. Vouloirs réaliser vite des vérifications formelles en mode « pompier » quand un projet échoue alors que cela n’a pas été prévu dès le départ ne marche que très rarement.*

*Malheureusement, il est difficile de convaincre les chefs de projet qu’il vaut mieux de travailler de cette façon préventive et incrémentale au lieu de se contenter de la méthode classique qui consiste à tout programmer et à tout tester après. Comme les coûts additionnels arrivent plus tôt, ils résistent à cette idée, souvent sans réaliser que le bénéfice sera bien plus considérable en fin de compte…*

*Mais on sait bien que même si bien des fumeurs sont persuadés qu’il serait bon qu’ils arrêtent, la plupart ne le font que lorsqu’ils craignent d’être vraiment malades, et recommencent quand les doutes s’estompent. Rien n’est plus difficile à vendre que la prévention…*

**Trois expériences personnelles intéressantes**

*Personnellement, j’ai toujours été passionné par les relations entre les mathématiques et la pratique du développement de systèmes informatisés…*

***Esterel et l’avionique***

*Au début des années 1990, je développais l’outillage de mon langage Esterel avec comme objectif principal la programmation des logiciels en temps réel…*

*J’ai rapidement réalisé à ma grande surprise que la bonne façon de comprendre la nature profonde d’Esterel était par le calcul booléen, bien mieux adapté que les techniques d’automates finis que j’utilisais depuis 1983…*

*L’environnement Esterel d’Ilog était alors utilisé par Dassault Aviation pour le développement des spécifications et des logiciels de son avion de chasse Rafale…Un des problèmes de Dassault était le besoin de vérification formelle pour des sous-systèmes critique. L’un d’entre-eux était le BTM, boitier de test et de maintenance, chargé des tests au sol et en plein vol de l’avion et de ses équipements. Ce boitier coordonnait une vingtaine de tâches de test, certaines au sol et d’autres en vol. plusieurs contraintes de sécurité étaient essentielles : il fallait par exemple montrer qu’il était impossible de déclencher en vol certains tests prévus pour le sol, comme l’essai de tous les ailerons en excursion maximale – ce qui aurait bien sûr provoqué un accident majeur…*

***Esterel V7 et la synthèse de circuits***

*En 2007-2008, j’étais directeur scientifique de la société Esterel Technologies. Mon centre d’intérêt principal était la synthèse efficace de circuits électroniques à partir de programmes de haut niveau, celle initiée chez Digital Equipment…*

*Nous travaillions alors avec une équipe de Texas instruments sur la réalisation d’un système de gestion des données d’images pour la télévision haute définition (TVHD) par téléphone, sujet nouveau à l’époque et fort difficile…*

*Selon la procédure en place à l’époque, le design obtenu fut envoyé en Inde pour être testé. L’équipe de test s’est plainte car elle ne trouvait pas de bugs, alors que ses supérieurs la jugeaient précisément selon le nombre de bugs trouvés ! Il n’a pas été facile de prouver au management que notre méthode était la bonne.*

***Corriger les bugs, mais recompiler***

*…Malheureusement, quand on veut corriger un bug de circuit, il n’est pas toujours possible de corriger le code source des programmes et de le recompiler comme on le fait avec un logiciel : il faudrait pour cela refaire quasiment tous les masques, avec un coût additionnel important.*

*La seule solution pratique est de « patcher » les masques à la main, en rajoutant des portes logiques et en ajoutant ou redirigeant des fils. On laisse toujours quelques pour cent de place libre dans un circuit initial pour pouvoir le faire sans tout chambouler… ».*

**Que retenir pour une éducation technologique par les professeurs et les élèves ?**

Dans ce résumé j’ai relevé les concepts, outils, méthodes et connaissances techniques associés et/ou discutés et/ou à discuter en tout ou partie lors d’un enseignement de la partie informatique en collège, ceci à l’intérieur du présent programme de technologie collège, voire dans un futur programme. Un certain nombre d’entre-eux figurent déjà dans le programme actuel.

C’est peut-être expliquer autrement que par des cours d’informatique, mais aussi avec de petits exemples concrets de programmation à partir de ceux cités par G.B.

Voici le relevé dans l’ordre des chapitres que nous venons de résumer et sans classement.

*Bug ; Trous de sécurité ; Génie logiciel ; Méthodes formelles ; Assertions logiques ; Assistant de preuve.*

**Dans le prochain résumé**

Le prochain document Gérard Berry pointe et exprime les éléments clés pour l’avenir à partir des développements des chapitres précédents, dont l’enseignement de l’informatique dans le système scolaire français., avec les mots clés suivants : *Ordinateur quantique ; Logiciels ouverts et fermés ; Apprentissage automatique ; Intelligence artificielle ; Enseignement de l’informatique.*

**Bibliographie et sitographie**

1 - BERRY, G. (2017*). L’hyperpuisssance de l’informatique. Algorithmes, données, machines, réseaux*. [Éditions Odile Jacob](https://www.odilejacob.fr/catalogue/sciences/mathematiques/hyperpuissance-de-l-informatique_9782738139535.php). 512 pages. 35,00 euros. ISBN 978-2-7381-3953-5

2 - RAK, I. [site personnel](http://techno-hadf.pagesperso-orange.fr/index.html)

3 - RAK, I. [site personnel](http://techno-hadf.pagesperso-orange.fr/index.html)

4 - [PAGESTEC](http://www.pagestec.org/spip/)

­5 - BERRY, G. (2017). Op cité, pages 291-344.

­6 - BERRY, G. (2017). Op cité, pages 345-390.