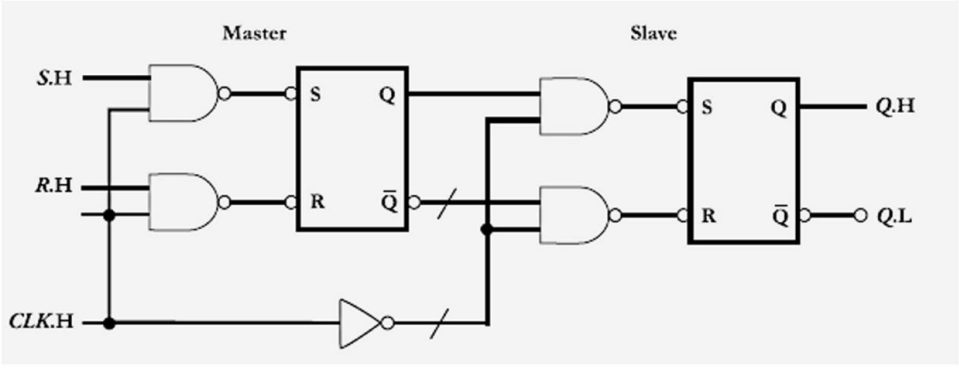


Bascule RS cadencée maître-esclave.

Sur les logiciels, ou la persistance des Connaissance visuelle

WENDY HUI KYONG CHUN

Lorsqu'un nombre suffisant de données apparemment insignifiantes sont analysées par rapport à des milliards d'éléments de données, l'invisible devient visible.
—Position 1

Jean Baudrillard dans L'Extase de la communication affirme que « nous ne participons plus au drame de l'aliénation, mais sommes dans l'extase de la communication. Et cette extase est obscène » parce que « dans la lumière brute et inexorable de l'information » tout est « immédiatement transparent, visible, exposé ».2 Bien qu'extrême, l'amalgame de Baudrillard entre l'information (et donc le calcul) et la transparence trouve un large écho dans l'opinion publique et politique. cercles universitaires, des craintes et de la propagande derrière les bases de données nationales aux examens de la « société de surveillance ». Cette confusion est remarquablement en contradiction avec les opérations informatiques réelles : pour que les ordinateurs deviennent des machines à transparence, le fait qu'ils calculent – qu'ils génèrent du texte et des images plutôt que de simplement représenter ou reproduire ce qui existe ailleurs – doit être oublié. Même lorsqu'ils sont attachés à des tubes de verre, les ordinateurs ne permettent pas simplement de voir ce qui se trouve de l'autre côté mais utilisent plutôt le verre pour envoyer et recevoir des impulsions lumineuses nécessaires pour recréer le référent (s'il en existe un). L'importance actuelle accordée à la transparence dans la conception des produits et dans le discours politique et scientifique est un geste compensatoire. À mesure que nos machines lisent et écrivent de plus en plus sans nous, à mesure que nos machines deviennent de plus en plus illisibles, de sorte que voir ne garantit plus de savoir (si jamais c'est le cas), nous, les soi-disant utilisateurs, nous voyons proposer davantage de voir, davantage de lecture. . L'ordinateur – cet appareil le moins visuel et le moins transparent – a paradoxalement favorisé la « culture visuelle » et la « transparence ».

Le logiciel – ou, pour être précis, la curieuse séparation entre le logiciel et le matériel – est à l'origine de ce geste compensatoire. Le logiciel perpétue certaines notions de voir comme savoir, de lecture et de lisibilité qui étaient censées s'estomper avec le déclin de l'indexicalité. Il le fait en imitant à la fois l'idéologie et la critique de l'idéologie, en confondant exécutable et exécution, programme et processus, ordre et action.3 Le logiciel, à travers des langages de programmation issus d'un système de commandement et de contrôle genré, discipline ses programmeurs et

utilisateurs, créant un système de visibilité invisible. Les connaissances proposées par les logiciels sont aussi obscures que révélatrices. Ainsi, si, comme le recommande Lev Manovich dans Language of New Media, les études sur les nouveaux médias doivent impliquer le logiciel, elles ne doivent pas simplement adopter le langage du logiciel, mais doivent examiner de manière critique les limites du « transcodage » et le nouveau statut du logiciel en tant que bon sens. 4

Matérialiser le logiciel immatériel est, ou devrait être, un concept notoirement difficile. La définition informatique actuelle du logiciel est un « ensemble d' instructions qui ordonnent à un ordinateur d'effectuer une tâche spécifique ». En tant qu'ensemble d'instructions, son statut matériel est instable ; en effet, plus vous disséquez un logiciel, plus il disparaît. L'historien Paul Ceruzzi le compare à un oignon, « avec de nombreuses couches distinctes de logiciels sur un noyau matériel ».5 Cette structure en forme d'oignon, cependant, est elle-même un effet de programmation : l'un code en utilisant un autre logiciel ; les logiciels et le matériel (comme les gènes et l'ADN) ne peuvent pas être physiquement séparés.

L'informaticien Manfred Broy décrit les logiciels comme « presque intangibles, généralement invisibles, complexes, vastes et difficiles à comprendre ». Parce que les logiciels sont « complexes, sujets aux erreurs et difficiles à visualiser », affirme Broy, nombre de ses « pionniers » ont cherché à rendre « les logiciels plus faciles à visualiser et à comprendre, à représenter les phénomènes rencontrés dans le développement logiciel dans des modèles qui rendent le logiciel plus facile à visualiser et à comprendre ». tâches d'ingénierie logicielle souvent implicites et intangibles explicites. »6 Friedrich Kittler a soutenu avec plus de force qu'« il n'y a pas de logiciel » puisque tout se réduit à des différences de tension en tant que signifiants.7 Dans les années 1940, le logiciel n'existait pas : il n'y avait littéralement pas de logiciel.8 « La programmation » comprenait la tâche humaine consistant à établir des connexions, à régler les commutateurs et à saisir des valeurs (« programmation directe »), ainsi que la tâche humaine et machine consistant à coordonner les différentes parties de l'ordinateur. En 1946, le programmeur principal de l'ENIAC (le premier ordinateur numérique électronique à usage général conçu, construit et utilisé avec succès) contrôlait la séquence d'actions nécessaires pour résoudre numériquement un problème.9 L'ENIAC a été initialement recâblé pour chaque problème de sorte que , essentiellement, un nouvel ENIAC était créé à chaque fois qu'il était utilisé. Sa conversion en ordinateur à programmes stockés en 1947 (en partie grâce à une suggestion de John von Neumann) signifiait que les programmes pouvaient être codés en réglant des commutateurs, qui correspondaient à soixante instructions stockées, plutôt qu'en branchant des câbles. Ce changement, considéré comme un moyen d'ouvrir la programmation aux simples scientifiques, a considérablement réduit le temps nécessaire à la programmation tout en augmentant le temps nécessaire au calcul. Aujourd'hui, ces paramètres modifiables seraient appelés logiciels car, avec les langages de programmation symbolique, ces paramètres physiques (qui permettaient par exemple de déplacer une valeur X de l'emplacement mémoire Y vers l'accumulateur) devenaient

traduit en une chaîne de nombres lus dans la mémoire de l'ordinateur.
Aujourd'hui, les « opérateurs » de bureau qui ont planifié et câblé l'ENIAC (Kathleen McNulty, Frances Bilas, Betty Jean Jennings, Elizabeth Snyder, Ruth Lichterman et Marlyn Wescoff) sont considérés comme parmi les premiers programmeurs.

Les langages de programmation symbolique et donc les logiciels, comme l'ont soutenu Paul Ceruzzi et Wolfgang Hagen, n'étaient pas prévus. L'émergence de la programmation en langage symbolique dépendait de la prise de conscience que l'ordinateur pouvait stocker des instructions numériques aussi facilement que des données et que l'ordinateur lui-même pouvait être utilisé pour traduire entre les notations symboliques et numériques. Les programmeurs de l'EDSAC, un des premiers ordinateurs (1949) de Cambridge, en Angleterre, ont été les premiers à utiliser l'ordinateur pour traduire entre un code assembleur plus lisible par l'homme (par exemple, « A100 » pour « ajouter le contenu de l'emplacement 100 à l'ajout). registre ») et ce qu'on appelle depuis langage machine (plutôt qu'un code logique). Le stockage a été la clé de l'émergence des langages de programmation, mais, comme le révèle le cas de John von Neumann, le stockage n'était pas suffisant : von Neumann, dont le nom est devenu le descripteur de tous les ordinateurs modernes à programmes stockés, a également conçu une notation similaire à l'EDSAC avec Herman Goldstine mais a supposé que les employés feraient la traduction.10 Un autre langage assembleur n'est pas un langage de programmation de niveau supérieur ; un ordinateur n'est pas automatiquement une machine multimédia. Selon Hagen, « pendant des décennies, l'arché-structure de la machine de von Neumann n'a pas révélé que cette machine serait plus qu'une nouvelle calculatrice, plus qu'un puissant outil de travail mental, à savoir un nouveau moyen de communication ». Le passage de la calculatrice au support de communication, affirme Hagen, découle lui-même d'un « impératif de communication » qui

> est né de la guerre froide, de l'économie, de l'organisation du travail, peut-être de la séduction numérique primitive exercée par les machines, du jeu des chiffres, d'un jeu avec des chiffres, des espaces réservés, des forts/da. mécanismes, et tout le quiproquo quasi-linguistique de la structure intérieure de toutes ces sources.11

Programmation automatique

La programmation automatique, ce que nous appelons aujourd'hui programmation, est née du désir de réutiliser le code et de recruter l'ordinateur dans son propre fonctionnement, c'est-à-dire de transformer des instructions singulières en un langage qu'un ordinateur pourrait écrire. Comme l'explique Mildred Koss, l'une des premières

> programmeuses d'UNIVAC : L'écriture du code machine impliquait plusieurs étapes fastidieuses : décomposer un processus en instructions discrètes, attribuer des emplacements de mémoire spécifiques à toutes les commandes et gérer les E/

> tampons. Après avoir suivi ces étapes pour mettre en œuvre les mathématiques routines, une bibliothèque de sous-programmes et des programmes de tri, notre tâche était d'examiner le processus de programmation plus large. Nous devions comprendre comment nous pourrions réutiliser le code testé et faire en sorte que la machine aide à la programmation. Pendant que nous programmions, nous avons examiné le processus et j'ai essayé de réfléchir à des moyens d'abstraire ces étapes pour incorporez-les dans un langage de niveau supérieur. Cela a conduit à développement d'interpréteurs, d'assembleurs, de compilateurs et de générateurs – des programmes conçus pour fonctionner sur ou produire d'autres programmes, c'est-à-dire de la programmation automatique.12

La programmation automatique est une abstraction qui permet la production de code informatique lisible par l'homme – clé de la marchandisation et de la matérialisation des logiciels et de l'émergence de langages de programmation de niveau supérieur. Cette automatisation de la programmation, en particulier des langages de programmation, rend la programmation problème plutôt que numérique. Les langages de programmation de niveau supérieur, contrairement au langage assembleur, font exploser les instructions et permettent d'oublier la machine. Ils permettent de gérer une programme sur plus d'une machine - une propriété maintenant supposée être une propriété « naturelle » du logiciel. La programmation directe » a conduit à un projet unique configuration des câbles; le premier langage machine pourrait être itérable mais uniquement sur la même machine - en supposant, bien sûr, aucune ingénierie défauts ou échecs. Pour émerger comme langage ou texte, les logiciels et les « langages » sur lesquels il s'appuie devaient devenir itérables. Avec langages de programmation, le produit de la programmation ne serait plus être une machine en marche, mais plutôt cette chose appelée logiciel, quelque chose théoriquement (sinon pratiquement) itérable, répétable, réutilisable, peu importe qui l'a écrit ou à quelle machine il était destiné. Les langages de programmation inscrivent l'absence à la fois du programmeur et de la machine. dans sa soi-disant écriture.13 Les langages de programmation ont permis de séparer l'instruction de la machine, l'impératif de l'action.

Selon une idée reçue, ces premières tentatives d'automatisation la programmation était inefficace et les « vrais » programmeurs résistaient. John Backus, développeur de FORTRAN, affirme que les premiers programmeurs de langage machine étaient engagés dans un « art noir » ; ils avaient un « une fierté chauvine dans leur sens des frontières et une conservatisme, tant de programmeurs des années 1950 en roue libre ont commencé se considérer comme membres d'une prêtrise gardant leurs compétences et des mystères bien trop complexes pour le commun des mortels. »14 Koss soutient de la même manière : « sans ces langages et processus de niveau supérieur. . . , qui a démocratisé la résolution de problèmes avec l'ordinateur, je crois que la programmation serait resté entre les mains d'un nombre relativement restreint de des rédacteurs de logiciels techniquement orientés utilisant du code machine, qui ont été essentiellement les grands prêtres de l'informatique. »15

La résistance à la programmation automatique semble également provenir des entreprises et des universités, pour qui les programmeurs coûtaient des ordres de grandeur moins chers à l'heure que les ordinateurs. Jean Sammett, l'une des premières programmeuses, raconte dans son ouvrage influent Programming Languages: History and Fundamentals,

> les clients ont soulevé de nombreuses objections, la principale étant que le compilateur ne pouvait probablement pas produire un code objet aussi bon que celui de leurs meilleurs programmeurs. Une importante campagne de vente visant à promouvoir les avantages de tels systèmes était en cours à cette époque, le fer de lance étant porté par les langages scientifiques numériques (c'est-à-dire FORTRAN) par IBM et par les langages de traitement de données commerciaux « de type anglais » par Remington Rand. (et le Dr Grace Hopper en particulier).16

Cette campagne de vente n'a pas seulement favorisé les langages de niveau supérieur (en dévalorisant les programmes produits par l'homme), elle a également favorisé l'apparition de nouveaux matériels : pour exécuter ces programmes, il fallait des machines plus puissantes. L'insistance du gouvernement sur la standardisation, particulièrement évidente dans le développement et la diffusion de COBOL, a également grandement influencé l'acceptation des langages de niveau supérieur, qui étaient là encore théoriquement , sinon toujours pratiquement, indépendants des machines ou Le cycle de mise à niveau du matériel a été normalisé au nom d'un gain de temps de programmation.

La « campagne de vente » a conduit à ce que beaucoup ont qualifié de démocratisation de la programmation. Pour Sammet, il s'agissait d'une révolution partielle,

> dans la manière dont les installations informatiques étaient gérées, car il devenait non seulement possible, mais tout à fait pratique, de permettre à des ingénieurs, des scientifiques et d'autres personnes de programmer réellement leurs propres problèmes sans l'intermédiaire d'un programmeur professionnel. Ainsi, le conflit entre l'atelier ouvert et l'atelier fermé est devenu très vif, souvent centré sur l'utilisation de FORTRAN comme illustration clé pour les deux côtés. Cela ne doit pas être interprété comme signifiant que toutes les personnes ayant des problèmes scientifiques numériques à résoudre se sont immédiatement mises à apprendre le FORTRAN ; ce n'est clairement pas vrai, mais un nombre si important d'entre eux l'ont fait que cela a eu un impact majeur sur l'ensemble de l'industrie informatique. L'un des effets secondaires subsidiaires de FORTRAN a été l'introduction du système de surveillance FORTRAN [IB60]. Cela a rendu l'installation informatique beaucoup plus efficace en nécessitant moins d'intervention de l'opérateur pour l'exécution du grand nombre de programmes FORTRAN (ainsi que le langage machine).17

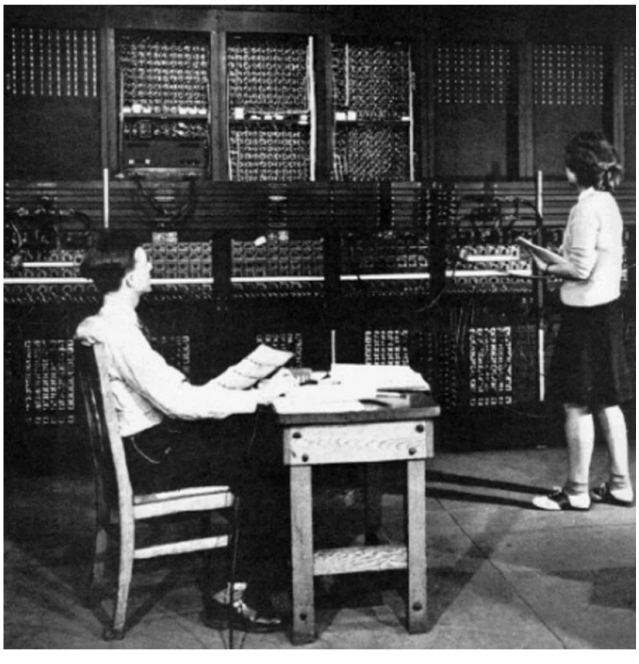
Cette « ouverture » de l'informatique, qui donne au terme open dans « open source » une résonance différente, signifierait la propagation potentielle de

l'informatique à ceux qui ont des problèmes numériques scientifiques à résoudre et le déplacement des opérateurs humains par les systèmes d'exploitation.
Mais les scientifiques ont toujours été impliqués dans l'informatique, même même si l'informatique n'a pas toujours été considérée comme un outil digne la poursuite scientifique et, comme mentionné précédemment, l'introduction de cadrans plutôt que de fils était censée responsabiliser les simples scientifiques. L'histoire de l'informatique est parsemée de moments de « libération informatique ».18

Ce récit de « l'ouverture » de l'informatique à travers des niveaux supérieurs langages suppose que les programmeurs appréciaient naturellement les tâches fastidieuses et tâches numériques répétitives et développer des solutions singulières pour leurs clients. La « maîtrise » de l'informatique peut facilement être comprise comme une « souffrance ». Harry Reed, l'un des premiers programmeurs d'ENIAC, affirme :

> L'idée même de l'informatique avec l'ENIAC était une sorte de
> une sorte de cilice . Programmer pour l'ordinateur, quel qu'il soit, était censé être une
> expérience rédemptrice - une
> était censé souffrir pour le faire. Et ce n'est que dans les années 1970
> que nous avons finalement réussi à convaincre les gens qu'ils n'étaient pas
> les programmeurs écriront continuellement de petits programmes
> pour eux. En fait, j'ai dû prendre ma division et asseoir tout le monde
> qui n'avait pas suivi de cours de FORTRAN, parce que, par Dieu,
> ils allaient écrire leurs propres programmes maintenant. Nous n'étions pas
> je vais demander à des informaticiens d'écrire des petits programmes simples
> qu'ils auraient dû écrire.19

Le récit de la démocratisation de la programmation révèle la tension au cœur des systèmes de programmation et de contrôle : contrôlent-ils systèmes ou servomécanismes (le nom initial de Norbert Wiener) ? La programmation est-elle une activité de bureau ou un acte de maîtrise ? Étant donné que le la machine s'occupe de la « programmation proprement dite » : la séquence des événements pendant l'exécution : la programmation est-elle réellement une programmation ? Qu'est-ce que compacté dans le passage linguistique de « opérateur » à « programmeur » ? La notion de « sacerdoce » de programmeurs efface cette tension, faisant de la programmation toujours déjà l'objet d'une tutelle jalouse, et effacer les fondements administratifs de la programmation. Programmation en les années 1950 semblent avoir été amusantes et assez équilibrées entre les sexes, en partie parce qu'elles étaient si nouvelles et en partie parce qu'elles n'étaient pas aussi lucratives comme la conception ou la vente de matériel informatique : la profession était neutre en termes de genre embaucher sinon payer parce que ce n'était pas encore une profession.20 L'« ENIAC filles » ont d'abord été embauchées comme sous-professionnelles, et certaines ont dû acquérir davantage de qualifications afin de conserver leur poste. Autant de femmes les programmeurs ont arrêté pour avoir des enfants ou se marier, les hommes ont pris leur des emplois de plus en plus lucratifs. Les fondements religieux et sans doute féminins de la programmation – tant en termes de personnel que de structure de commandement – ont été enterrés alors que la programmation cherchait à devenir un



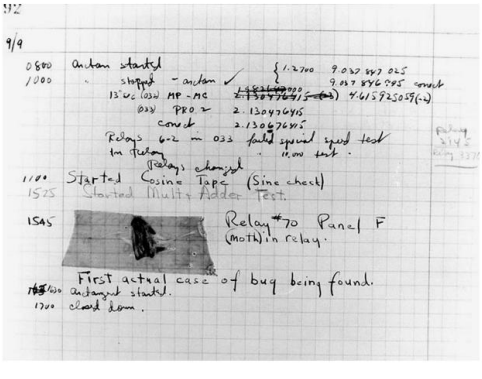
Programmeurs ENIAC, fin
des années 1940. Photo militaire américaine,
Archives de l'Arsenal de Redstone,
Huntsville, Alabama.

domaine de l'ingénierie et du monde universitaire dans son
propre droit. Un tel effacement est la clé du
professionnalisation de la programmation...
une maîtrise compensatoire fondée sur la dissimulation
la machine. La démocratisation n'a pas eu lieu
déplacer les programmeurs professionnels mais
ont plutôt renforcé leur position de professionnels en
diminuant paradoxalement
leur véritable pouvoir sur leurs machines
et en généralisant l'ingénierie
notion d'information.

Oui Monsieur

L'hypothèse selon laquelle les programmeurs aiment
naturellement les tâches fastidieuses souligne l'importance de la programmation et de l'informatique.
histoire genrée et humaine. Pendant la Seconde Guerre mondiale, presque tous les informaticiens
étaient des jeunes femmes ayant une certaine formation en mathématiques.
Non seulement les femmes étaient alors disponibles pour travailler, mais elles étaient également
considérées comme de meilleurs ordinateurs, plus consciencieux, probablement parce que
ils étaient meilleurs dans les tâches répétitives et de bureau. Les programmeurs étaient d'anciens
ordinateurs car ils étaient les mieux adaptés pour préparer leurs successeurs :
ils pensaient comme des ordinateurs.

On pourrait dire que la programmation est devenue programmation et que le logiciel est
devenu logiciel lorsque les commandes sont passées du commandement d'un
« fille » pour commander une machine. L'image ci-dessus révèle le rêve
de « programmation proprement dite » : un homme assis à un bureau donnant des commandes
à une « opératrice » féminine. Les langages logiciels sont basés sur une série de
impératifs découlant de la structure de commandement et de contrôle de la Seconde Guerre
mondiale. L'automatisation du commandement et du contrôle, que Paul Edwards
a identifié comme une perversion des traditions militaires de « leadership personnel, de
commandement décentralisé sur le champ de bataille et de
autorité21 », a sans doute commencé avec le calcul mécanique de la Seconde Guerre mondiale .
Ceci est illustré de manière plus frappante par la relation entre
les Wrens, membres bénévoles du Women's Royal Naval Service,
et leurs commandants à Bletchley Park. Les Wrens, aussi
appelés esclaves par Turing (un terme désormais intégré aux systèmes informatiques), étaient
des employés responsables du fonctionnement mécanique de l'ordinateur.
machines de cryptanalyse (la Bombe puis le Colossus), bien que
au moins l'une des commis, Joan Clarke, est devenue analyste. De manière révélatrice,
IJ Good, un analyste masculin, décrit le Colossus comme permettant une synergie homme-
machine reproduite par les machines modernes seulement à la fin.
Années 1970 : « l'analyste s'asseyait devant la sortie de la machine à écrire et criait
instructions à un Wren pour apporter des modifications aux programmes. Certains
d'autres utilisations ont finalement été réduites à des arbres de décision et ont été confiées
aux opérateurs de machines (Wrens). »22 Cette synergie homme-machine,



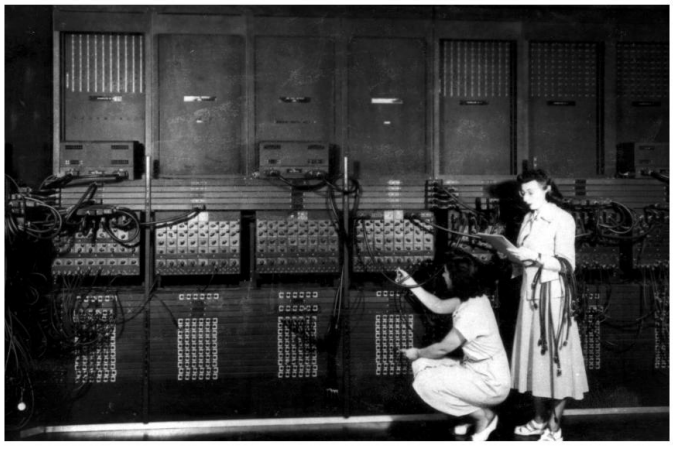
ou un traitement interactif en temps réel (plutôt que par lots), traitant les Wrens et les machines de manière indiscernable, tout en s'appuyant simultanément sur la capacité des Wrens à répondre aux ordres du mathématicien.

L'histoire de la première rencontre entre Grace Murray Hopper (l'une des premières et des plus importantes programmeuses) et Howard Aiken conforte également ce récit. Hopper, titulaire d'un doctorat. en mathématiques de Yale et ancien professeur de mathématiques à Vassar, a été chargé par l'US Navy de programmer le Mark I, un ordinateur numérique électromécanique qui faisait le bruit d'une pièce remplie d'aiguilles à tricoter. D'après Hopper, Aiken lui a montré

> un grand objet à trois rayures. . . a agité la main et a dit : « C'est une machine informatique. » J'ai dit: "Oui, monsieur." Que pourrais-je dire d'autre ? Il a dit qu'il aimerait que je calcule les coefficients de la série arc-tangente, pour jeudi. Encore une fois, que pourrais-je dire ? "Oui Monsieur." Je ne savais pas ce qui se passait , mais c'était ma rencontre avec Howard Hathaway Aiken.23

Le calcul dépend du « oui, monsieur » en réponse à de courtes phrases déclaratives et à des impératifs qui sont essentiellement des commandes. Contrairement à Neal Stephenson, au début il y avait la commande plutôt que la ligne de commande. La ligne de commande est une simple simulation du système d'exploitation (OS). Les commandes ont permis le glissement entre la programmation et l'action qui fait du logiciel un système de communication si convaincant mais logiquement « trivial ». Les souvenirs d'IJ Good et Hopper révèlent également la routinisation au cœur de la programmation : l'analyste de Bletchley Park fut bientôt remplacé par des arbres de décision. Hopper, le programmeur, est devenu un partisan de la programmation automatique. Ainsi, la routinisation ou l'automatisation est au cœur d'une profession qui aime croire qu'elle a réussi à automatiser toutes les professions sauf la sienne24.

Mais considérer les femmes et le calcul mécanique comme interchangeables revient à réviser l'histoire. Selon Sadie Plant, la programmation est essentiellement féminine, non seulement parce que les femmes, d'Ada Lovelace à Hopper, ont été les premières programmeuses, mais en raison des liens historiques et théoriques entre la programmation et ce que Freud a appelé l' invention typiquement féminine du tissage, entre la sexualité féminine. comme le mimétisme et le mimétisme qui fonde la vision de Turing des ordinateurs en tant que machines universelles. (De plus, les logiciels et la sexualité féminine révèlent le pouvoir que peut avoir quelque chose qui ne peut pas être vu.)25 Les femmes, affirme Plant,



n'ont pas simplement eu un rôle mineur
jouer dans l'émergence des machines
numériques. . . . Le leur n'est pas un
rôle subsidiaire qui doit être
sauvé pour la postérité, un petit
supplément dont l'inclusion
remettre les pendules à l'heure
. . . . Matériel, logiciels, logiciels – avant leurs débuts et
au-delà de leurs fins, les femmes ont été des simulateurs, des assembleuses,
et les programmeurs des machines numériques.26

La photographie de la page 33 n'est pas représentative : les programmeuses de l'ENIAC travaillaient ensemble par paires, et aucune machine aurait pu accomplir ce que Hopper a fait – du moins pas à ce moment-là. (Et encore une fois, Hopper serait la clé pour permettre aux ordinateurs de le faire : la réduction de la distance entre Hopper et les ordinateurs serait la clé pour y parvenir. commande et contrôle automatique). La difficulté rencontrée par les programmeurs était simple : les ordinateurs n'étaient pas des ordinateurs. Le passage de commander une fille à commander un automate était difficile parce que les automates déchiffraient plutôt qu'interprétaient ou présumaient, et qu'ils n'apprenaient pas de l'expérience. Comme Martin Selon Campbell-Kelly et William Aspray, « la difficulté fondamentale de l'écriture de logiciels résidait dans le fait que, jusqu'à l'arrivée des ordinateurs, les êtres humains les êtres n'avaient jamais eu à préparer d'instructions détaillées pour un automate : une machine qui obéit infailliblement aux commandes données à cela, et pour lequel tous les résultats possibles devaient être anticipés par le programmeur. »27 L'évaluation de Campbell-Kelly et Aspray surestime la fiabilité des machines, en particulier les premières. Comme Comme le racontent les premiers programmeurs ENIAC, une partie du débogage consistait à déterminer quelles erreurs provenaient de la programmation et lesquelles provenaient d'un défaut tubes à vide, recâblage accidentel ou défauts dans l'architecture des machines, une tâche facilitée par les ampoules au néon fixées sur divers compteurs.

Ci-contre, en haut : Capitaine Grace M. Hopper, 1976. Marine américaine Photo (numéro NH 96945), Centre historique naval, Washington DC

Ci-contre, en bas : Premier « Bug » informatique, 1945. Photo de l'US Navy (numéro NH 96566-KN), historique naval Centre, Washington, DC

Ci-dessus : Deux femmes connectant le côté droit de l'ENIAC avec un nouveau programme, fin des années 1940. Photo de l'armée américaine, Armée Laboratoire de Recherche Technique Bibliothèque, Aberdeen Prouvant Terrain, Aberdeen, Maryland.

Contrairement aux machines, les programmeuses ne se contentaient pas de suivre instructions. Hopper a été décrite comme « une femme dotée d'une forte personnalité, une puissante persuasive et une leader. Elle a montré une partie de sa Marine formation à son discours imposant. »28 De plus, la programmation, comme le dit Koss explique, il ne s'agissait pas seulement de mettre en œuvre des instructions mais de « concevoir un stratégie et préparer des instructions pour que l'ordinateur fasse ce que vous Je voulais que ce soit pour résoudre un problème. »29 Les programmeuses ont joué un rôle rôle important dans la conversion de l'ENIAC en un ordinateur à programme stocké et pour déterminer le compromis entre le stockage des valeurs et des instructions. Betty Snyder Holberton, décrite par Hopper comme la meilleure programmeuse qu'elle ait connue, n'a pas seulement débogué la trajectoire balistique programme dans un rêve (le premier programme à être exécuté sur l'ENIAC, bien que trop tard pour être utile à la Seconde Guerre mondiale), elle a également développé un type influent

algorithme pour l'UNIVAC.30

L'ampleur de la répression contre ces femmes
dans les histoires standards de l'informatique peut être
mesurée par la théorie implicite et a priori de Paul Edwards.
genre de la « force de travail » informatique dans un
une analyse autrement perspicace de la masculinité
et la programmation. Il écrit:

> les informaticiens jouissent d'une mystique de maîtrise acharnée comparable au
> culte des physiciens dans les années d'après-guerre. Des ordinateurs
> leur fournir une précision sans faille, une puissance de calcul,
> et la capacité de synthétiser d'énormes quantités de données. . . .
>
> Il n'y a rien de fondamentalement masculin dans la technologie informatique.
> Autrement, les femmes n'auraient pas pu entrer aussi rapidement dans le monde
> de l'informatique. Les valeurs de genre sont largement
> flottent librement des machines elles-mêmes et sont exprimées et
> renforcée par les relations de pouvoir entre hommes et femmes.
> Les ordinateurs n'incarnent pas simplement la masculinité ; ils sont culturellement
> construits comme des objets mentaux masculins.31

On est loin des affirmations de J. Chuan Chu (l'un des premiers auteurs d'ENIAC
ingénieurs matériels) que le logiciel est la fille de Frankenstein
(le matériel étant son fils), l'évaluation d'Edwards efface la présence écrasante des
femmes dans les débuts de l'informatique – leur travail en tant qu'êtres humains.
ordinateurs, programmeurs et moniteurs, tout en réduisant les ordinateurs
de la technologie au logiciel. Comme la combinaison d'un employé humain et d'un
ordinateur humain, l'ordinateur moderne résume les relations de pouvoir entre les
hommes et les femmes dans les années 1940. Il cherchait à déplacer
les femmes : leurs doigts agiles, leurs capacités numériques, leur discrétion , leurs
« regards inquiétants » – un déplacement perçu par Vannevar Bush
comme souhaitable.32 La transition des ordinateurs humains aux ordinateurs mécaniques
relations de pouvoir différentielles automatisées.

Reconnaître ces femmes comme programmeuses – ce n'est pas simplement suivre
mais aussi rédiger des instructions - est important mais pas suffisant, car
il maintient en place le récit de la programmation comme étant « magistral ». Qu'est-ce que
l'importance de suivre et de mettre en œuvre les instructions ? Peut-être
l'« automatisation » du contrôle et du commandement est moins une perversion de la
tradition militaire qu'une instanciation de celle-ci, dans laquelle la responsabilité a été
transférée à ceux (désormais des machines) qui mettent en œuvre la tradition militaire.
commandes. La relation entre maîtres et esclaves est toujours
ambiguë. Cette transmission du pouvoir a été masquée par des langages de
programmation qui occultent la machine et mettent en avant la programmation (plutôt que
l'exécution) comme source d'action. Le
clôture de la distinction entre programmation et exécution, attestée par l'ambiguïté de
l'objet du verbe « programmer »,
a été facilitée par la discipline et la professionnalisation des pro-

Les quatre premiers, années 1950. L'armée américaine
Photo (numéro 163-12-62).

programmeurs grâce à la « programmation structurée ».

Cacher la machine

Durant la « crise logicielle » tant discutée de la fin des années 1960, qui découlait de débâcles aussi spectaculaires que l'OS/360 d'IBM, de nombreux (en particulier des programmeurs européens, comme Friedrich [Fritz] Bauer et Peter Naur) considérait le « génie logiciel », ou programmation structurée, comme un moyen de faire passer la programmation d'une pratique artisanale à une pratique industrielle standardisée, et comme un moyen de créer des programmeurs disciplinés qui s'occupaient d'abstractions plutôt que de données numériques. processus33. Comme l'a soutenu Michael Mahoney, la programmation structurée est apparue comme un « moyen à la fois de contrôle de la qualité et de discipline ». programmeurs, méthodes de comptabilité analytique et d'estimation, méthodes de vérification et de validation, techniques d'assurance qualité. »34

La « programmation structurée » (aussi généralement appelée « bonne programmation ») cache, et donc sécurise, la machine. Sans surprise, avoir peu ou pas de contact avec la machine réelle améliore capacité à penser de manière abstraite plutôt que numérique. Edsger Dijkstra, dont la célèbre condamnation des déclarations « aller à » a résumé à de nombreux principes fondamentaux de la programmation structurelle, estime qu'il était capable de « pionnier » avec précision dans la programmation structurée parce qu'il a commencé sa carrière de programmeur en codant pour des machines cela n'existait pas encore.35 Dans « Aller à la déclaration considérée comme nuisible », Dijkstra affirme que « la qualité des programmeurs est une fonction décroissante ». de la densité des déclarations go to dans les programmes qu'ils produisent. Ce c'est parce que les déclarations vont à l'encontre du principe fondamental du bien programmation – la nécessité de « réduire l'écart conceptuel entre programme statique et processus dynamique, pour faire la correspondance entre le programme (étalé dans l'espace texte) et le processus (étalé dans le temps) aussi trivial que possible. Plus précisément, si un programme est arrêté, les gotos rendent difficile la recherche d'une place dans la programmation qui correspond au processus arrêté – cela le rend « terriblement difficile de trouver un ensemble significatif de coordonnées permettant de décrire le progrès du processus. »36 Autrement dit, les allers-retours rendent difficile l'amalgame des instruction avec commandement, qui fonde la « programmation ».37

Les langages de programmation structurés « sauvent » les programmeurs eux-mêmes en assurant une bonne sécurité, où sécurité signifie sécurité du programmeur.38 Au nom de la sécurité, la programmation structurée, qui met l'accent sur la programmation comme une question de flux, est elle-même cédant la place à l'abstraction des données, qui considère la programmation comme une question d'objets interdépendants et cache bien plus que la machine. Données l'abstraction dépend de la dissimulation de l'information, de la non-réflexion de faits modifiables dans le logiciel. Comme John V. Guttag, un « pionnier » des données l'abstraction explique, l'abstraction des données consiste à oublier39 .
que de « polluer » un programme en permettant des lignes de contact invisibles

entre des modules supposés indépendants, l'abstraction de données présente une interface propre ou « belle » en confinant les spécificités, et en réduisant les connaissances et le pouvoir du programmeur. La connaissance, insiste Guttag, est dangereuse : « « Boire profondément ou ne pas goûter à la source de Piérie » n'est pas nécessairement un bon conseil. En savoir trop n'est pas mieux, et souvent pire, que d'en savoir trop peu. Les gens ne peuvent pas assimiler beaucoup d'informations. Toute méthode ou approche de programmation qui suppose que les gens comprendront beaucoup de choses est très risquée. »40

Ainsi, l'abstraction donne du pouvoir au programmeur et insiste sur son ignorance. Parce que l'abstraction existe « dans l'esprit du programmeur », l'abstraction donne aux programmeurs de nouvelles capacités créatives.
L'informaticien David Eck affirme que « chaque langage de programmation définit une machine virtuelle, dont il est le langage machine.
Les concepteurs de langages de programmation créent des machines informatiques aussi sûrement que l'ingénieur qui travaille dans le silicium et le cuivre, mais sans les limitations imposées par les matériaux et la technologie de fabrication. »41
Cependant, cette abstraction, cet éloignement des spécificités du machine – cède, dans sa séparation de la machine en logiciel et matériel, l'acte de programmation à la machine elle-même. Koss s'est moqué de la première notion selon laquelle les ordinateurs étaient des cerveaux, car « ils ne pouvaient pas penser de la même manière qu'un humain, mais devaient recevoir un ensemble d'instructions machine étape par étape à exécuter avant de pouvoir fournir des réponses à une question spécifique ». problème » – à cette époque, le logiciel n'était pas considéré comme un objet indépendant.42 Le statut actuel du logiciel en tant que marchandise, malgré le fait que ses instructions soient immatérielles et non rivales, indique le triomphe de l'industrie du logiciel, une industrie qui au début luttait pour ne pas seulement financièrement mais aussi conceptuellement pour définir son produit. L'essor du logiciel dépend à la fois de mouvements historiques, tels que la séparation par IBM de ses services de ses produits, et des abstractions permises par les langages de niveau supérieur.
L'insistance de Guttag sur le manque de fiabilité et l'incapacité des êtres humains à comprendre souligne les coûts d'une telle abstraction.
L'abstraction est le jeu de l'ordinateur, tout comme la programmation au sens strict du terme.

Il est important de noter que les programmeurs sont des utilisateurs : ils créent des programmes à l'aide d'éditeurs, qui sont eux-mêmes des logiciels. La distinction entre programmeurs et utilisateurs s'estompe progressivement, non seulement parce que les utilisateurs deviennent des programmeurs (au sens propre, les programmeurs ne programment plus un ordinateur ; ils codent), mais aussi parce qu'avec les langages de haut niveau, les programmeurs ressemblent davantage à de simples utilisateurs. .
La différence entre les utilisateurs et les programmeurs est un effet du logiciel.

Plaisir causal La

rétrogradation progressive des programmeurs a été compensée par le pouvoir et le plaisir de la programmation. Comme le soutient Edwards, « la programmation peut

produire de fortes sensations de pouvoir et de contrôle » parce que l'ordinateur produit un micromonde cohérent en interne bien qu'incomplet en externe ,

> un monde simulé, entièrement au sein de la machine elle-même, qui ne dépend pas de l'efficacité instrumentale. Autrement dit, là où la plupart des outils produisent des effets sur un monde plus vaste dont ils ne sont qu'une partie, l'ordinateur contient ses propres mondes en miniature. . .
> Dans le micromonde, comme dans les fictions des enfants, le pouvoir du programmeur est absolu.43

Ce plaisir est lui-même un effet des langages de programmation, qui offrent l'attrait de la visibilité, de la lisibilité, de la cause et de l'effet. Considérez ce programme omniprésent « hello world » écrit en C++ (« hello world » est généralement le premier programme qu'une personne écrira) :

```
// ce programme crache "hello world" #include
<iostream.h> int main() {


    cout << « Bonjour tout le monde ! » ;
    renvoie 0 ;
}
```

La première ligne est une ligne de commentaire, expliquant au lecteur humain que ce programme crache « bonjour tout le monde ». La ligne suivante demande au préprocesseur du compilateur d'inclure iostream.h, un fichier standard pour gérer les entrées et les sorties. La troisième ligne, « int main() », commence la fonction principale du programme ; « cout<<« Bonjour tout le monde ! » ; » affiche « Hello World » à l'écran (« cout » est défini dans iostream.h); « return 0 » termine la fonction principale et amène le programme à renvoyer un 0 s'il s'est exécuté correctement. Bien que pas immédiatement compréhensible pour quelqu'un qui n'est pas versé en C++, ce programme semble néanmoins avoir un certain sens. Il comprend une série d'impératifs et de déclaratifs que l'ordinateur comprend et respecte vraisemblablement. Lorsqu'il s'exécute, il suit les commandes et affiche « Hello World ». Surtout, ce message, qui affirme l' agence du programmeur, la remet également en question : qui ou quoi, après tout, dit « bonjour tout le monde ? Pour bénéficier de ce pouvoir absolu, le programmeur doit suivre les règles d'un langage de programmation. Quoi qu'il en soit, voir son code produire des résultats visibles et largement prévisibles crée du plaisir44. Son code provoque la réalisation d'une action : la cause et l'effet sont clairs, même si le résultat final n'est jamais entièrement prévisible. Ce pouvoir absolu rendu possible grâce à l'action d'un programme révèle le statut contradictoire de l'action, à savoir le fait que l'action se réfère à la fois à la capacité d'une personne à agir et à la capacité de quelqu'un d'autre à agir en son nom.

Cependant, dans le micromonde de la simulation informatique, il ne s'agit pas seulement

le programmeur dont le pouvoir est amélioré ou absolu, car les simulations interactives - clé du concept d'ordinateurs transparents - améliorent le pouvoir de l'utilisateur (encore une fois, ces termes ne sont pas absolus mais dépendent plutôt du logiciel utilisé et de la pouvoir productif de ses actions). L'interactivité, intimement liée, comme l'a soutenu Edwards, à l'intelligence artificielle, découlait initialement de l'admission de la faillibilité humaine et des limites des langages de programmation procédurale45 . Dans les années 1960, la naïveté derrière l'affirmation de von Neumann selon laquelle « tout ce qui est qui peut être décrit de manière exhaustive et sans ambiguïté, tout ce qui peut être mis en mots complètement et sans ambiguïté est ipso facto réalisable par un réseau neuronal fini approprié » devenait de plus en plus évident.46 Étant donné qu'une description exhaustive et sans ambiguïté était difficile, voire impossible, Travailler « de manière interactive » avec un ordinateur pour résoudre des problèmes était essentiel. L'interactivité a ensuite été confondue avec la liberté de l'utilisateur et le contrôle avec les interfaces GUI (utilisateur graphique) et WYSIWYG (What You See Is What You Get), qui étaient considérées comme des compléments aux commandes basées sur le langage. Contrairement aux interfaces de ligne de commande, les interfaces graphiques permettaient une « manipulation directe », dont la maîtrise était intimement liée à la visibilité simulée. Selon Ben Schneiderman :

> Certains systèmes interactifs suscitent un enthousiasme enthousiaste parmi les utilisateurs, en contraste frappant avec la réaction plus courante d' acceptation réticente ou d'hostilité pure et simple. Les rapports des utilisateurs enthousiastes sont remplis de sentiments positifs concernant :
>
> • maîtrise du système • compétence dans l'exécution de leur tâche • facilité à apprendre le système de manière originale et à assimiler des fonctionnalités avancées
>
> • confiance dans leur capacité à conserver la maîtrise dans le temps • plaisir à utiliser le système
>
> • désir de le montrer aux novices et • désir d'explorer des aspects plus puissants du système
>
> Ces sentiments ne sont bien sûr pas universels, mais l'amalgame véhicule l'image d'un utilisateur véritablement satisfait. . . . Les idées centrales semblaient être la visibilité de l'objet d'intérêt ; actions rapides, réversibles et incrémentielles et remplacement de la syntaxe complexe du langage de commande par une manipulation directe de l'objet d'intérêt – d'où le terme « manipulation directe ».47

Comme Brenda Laurel l'a soutenu dans sa comparaison des interfaces informatiques et du théâtre, la manipulation directe (qui est tout sauf directe) doit être complétée par un engagement direct pour réussir.
Engagement direct, affirme Laurel

> détourne l'attention de la représentation d'objets manipulables
> à l'idéal de permettre aux gens de s'engager directement dans l'activité
> choix, qu'il s'agisse de manipuler des outils symboliques dans l'exécution de
> certaines tâches instrumentales ou de se promener dans le
> monde imaginaire d'un jeu vidéo. L'engagement direct met l'accent sur
> les valeurs émotionnelles et cognitives. Il conçoit
> l'activité homme-machine en tant qu'expérience conçue48.

L'accent mis par Laurel sur l'action souligne la différence cruciale entre la représentation des outils et les outils eux-mêmes : elle soutient que les gens se rendent compte lorsqu'ils double-cliquent sur un dossier qu'il s'agit ce n'est pas vraiment un dossier, et rendre un dossier plus « réaliste » n'est pas utile. Ce qui est utile, affirme Laurel, c'est une causalité claire : les événements doivent se produire de telle manière que l'utilisateur puisse les accepter comme probables et donc réduire la probabilité à la certitude. La causalité, affirme-t-elle, garantit l'universalité, garantit que les utilisateurs suspendront volontairement leur incrédulité. Pour les utilisateurs comme pour les schizophrènes paranoïaques (mon observation, pas Laurel's), tout a un sens : il ne peut y avoir de coïncidences, seulement plaisir causal.

Le plaisir causal n'est pas simplement une représentation des actions de l'utilisateur dans un manière causalement plausible ; c'est aussi une « amplification utilisateur ». Manovitch explique « l'amplification de l'utilisateur » en termes de Super Mario :

> lorsque vous dites à Mario de faire un pas vers la gauche en déplaçant un joystick, cela
> initie un petit récit délicieux : Mario traverse une colline ;
> il commence à gravir la colline ; la colline s'avère trop raide ;
> Mario retombe sur le sol ; Mario se lève, tout tremblant.
> Aucune de ces actions n'exigeait quoi que ce soit de notre part ; tout ce qu'il fallait
> faire est simplement de déplacer le joystick une fois. Le programme informatique
> amplifie notre action unique, l'étendant en une séquence narrative.49

Cette amplification utilisateur imite la conduite en « explosion d'instructions » langages de programmation de niveau supérieur (une ligne de code de haut niveau correspond à plus d'une ligne de code machine) ; amplification utilisateur n'est pas seulement le produit de logiciels de jeu et de l'art logiciel, mais il est au cœur du pouvoir de la programmation.

Cette double amplification est sans doute à l'origine de la romantisation de programmation et, plus récemment, l'émergence de l'art logiciel, ou Génération Flash. Selon Manovich, la génération Flash est une nouvelle groupe d'artistes (« nouveaux romantiques ») qui créent un code original plutôt que de participer au cycle sans fin des citations postmodernes. Comme le produisent les artistes de Generation Flash

> le nouveau modernisme des visualisations de données, des réseaux vectoriels, minces en pixels
> grilles et flèches : le design Bauhaus au service de l'information
> conception. Au lieu de l'assaut baroque des médias commerciaux, Flash

> Cette génération nous sert l'esthétique moderniste et la rationalité du logiciel. La conception de l'information est utilisée comme un outil pour donner un sens à la réalité tandis que la programmation devient un outil d'autonomisation.50

Pour donner un sens à la réalité, ces artistes et designers utilisent l'amplification de l'utilisateur, car leur esthétique moderniste et leur rationalité logicielle amplifient et simplifient les causes et les effets. Dans la lignée des logiciels en général, ils dévoilent, ils rendent les choses visibles. Ce dévoilement dépend d'une certaine « intelligence » de la part de l'utilisateur. Décrivant le projet de Futurefarmer, Theyrule.net, qui offre aux utilisateurs un moyen de cartographier les relations entre les membres des conseils d'administration des entreprises les plus puissantes, Manovich déclare :

> Au lieu de présenter un message politique emballé, il nous fournit des données et les outils nécessaires pour les analyser. Il sait que nous sommes suffisamment intelligents pour tirer la bonne conclusion. C'est la nouvelle rhétorique de l'interactivité : on se laisse convaincre non pas en écoutant/ regardant un message préparé mais en travaillant activement avec les données : en les réorganisant, en découvrant les liens, en prenant conscience

des corrélations51. Selon Manovich, cette nouvelle rhétorique de l'interactivité l'interactivité est explorée plus en détail dans UTOPIA :

> La cosmogonie de ce monde reflète notre nouvelle compréhension de notre propre planète : l'après-guerre froide, Internet, l'écologie, Gaia et la mondialisation. Remarquez les fines lignes à peine visibles qui relient les acteurs et les blocs. (Il s'agit du même dispositif utilisé dans Theirrule.net.) Dans l'univers d'UTOPIA, tout est interconnecté et chaque action d'un acteur individuel affecte le système dans son ensemble. Intellectuellement, nous savons que c'est ainsi que notre Terre fonctionne écologiquement et économiquement – mais UTOPIE représente cela à une échelle que nous pouvons saisir perceptuellement.52

L'UTOPIE permet ce que Fredric Jameson a appelé une « carte cognitive » : « une représentation situationnelle de la part de l'individu soumis à cette totalité plus vaste et proprement irreprésentable qui est l'ensemble des structures de la société dans son ensemble. »53 Si la cartographie cognitive est À la fois difficile et nécessaire aujourd'hui en raison des réseaux invisibles du capital, ces artistes produisent une carte cognitive en exploitant l'invisibilité de l'information. Le fonctionnement de l'art logiciel, comme le soutient Manovich, est parallèle à la critique de l'idéologie marxiste. Le voile de l'idéologie est déchiré par la compréhension des relations entre l'action des acteurs individuels et le système dans son ensemble. Le logiciel permet cette critique en la représentant à une échelle – dans un micromonde – que nous pouvons comprendre.
Ce dévoilement dépend de nos propres actions, de notre manipulation des objets pour voir, de notre réflexion comme des programmeurs orientés objet.
Plutôt que de manquer de cartes cognitives, nous en produisons tout le temps

à travers un médium qui simule la critique de l'idéologie et, dans sa non-existence, l'idéologie également. Il est vraiment remarquable que les logiciels... conçu pour obscurcir la machine et en créer une virtuelle et basé sur sur des commandes enterrées – a conduit à la notion écrasante de calcul transparent. Cette notion de transparence a moins à voir avec opérations technologiques réelles qu'avec le « micromonde » établi par le calcul.

Le logiciel comme idéologie

Comme je l'ai soutenu ailleurs, le logiciel est un analogue fonctionnel de l'idéologie.54 Au sens formel , les ordinateurs sont compris comme comprenant des logiciels et le matériel est des machines à idéologie. Ils remplissent presque toutes les formalités définition de l'idéologie que nous avons, de l'idéologie comme fausse conscience (telle que décrite dans Matrix) à la définition de l'idéologie donnée par Louis Althusser comme « une « représentation » de la relation imaginaire des individus à leurs conditions réelles d'existence. »55 Les logiciels, ou peut-être plus précisément les systèmes d'exploitation, nous proposent une relation imaginaire avec nos matériel : ils ne représentent pas des transistors mais plutôt des ordinateurs de bureau et bacs de recyclage. Le logiciel produit des « utilisateurs ». Sans OS, il y aurait n'avoir aucun accès au matériel ; sans système d'exploitation, aucune action, aucune pratique et donc aucun utilisateur. Chaque OS, à travers ses publicités, interpelle un « utilisateur » : l'appelle et lui propose un nom ou une image avec laquelle s'identifier. Alors Mac les utilisateurs « pensent différemment » et s'identifient à Martin Luther King et Albert Einstein; Les utilisateurs de Linux sont des passionnés de l'énergie open source, attirés par l'image d'un pingouin gros et repu ; et les utilisateurs de Windows sont des types fonctionnalistes traditionnels, peut-être réconfortés, comme le soutient Eben Moglen : par leurs ordinateurs qui plantent régulièrement. Surtout, les « choix » Les systèmes d'exploitation offrent une limite au visible et à l'invisible, à l'imaginable et à l'inimaginable. Cependant, vous n'êtes pas conscient de la restriction et de l'interpellation constantes du logiciel (également appelées son "convivialité"), à moins que vous ne soyez frustré par ses paramètres par défaut (qui sont remarquablement appelés vos préférences) ou vous utilisez plusieurs systèmes d'exploitation ou progiciels concurrents.

Le logiciel produit également des utilisateurs grâce à des interactions bénignes, depuis des sons rassurants qui signifient qu'un fichier a été enregistré dans des noms de dossiers comme « mes documents », qui mettent l'accent sur la possession d'un ordinateur personnel. Les programmes informatiques utilisent sans vergogne des métamorphes, des pronoms comme « mon » et « vous » qui s'adressent à vous, ainsi qu'à tous les autres, en tant que sujet. Un logiciel vous fait lire, vous offre plus de relations et toujours plus de visuels. Les logiciels provoquent des lectures qui vont au-delà de la lecture de lettres vers des pratiques non littéraires et archaïques de deviner, interpréter, compter et répéter. Le logiciel est basé sur un fétichisme logic.56 Les utilisateurs savent très bien que leurs dossiers et bureaux ne sont pas vraiment des dossiers et des bureaux, mais ils les traitent comme s'ils l'étaient - en en les désignant comme des dossiers et des bureaux. Cette logique est, selon

à Slavoj Zizek, crucial pour l'idéologie. Zizek (par l'intermédiaire de Peter Sloterdjik) soutient que l'idéologie persiste dans les actions plutôt que dans les croyances. L'illusion de l'idéologie n'existe pas au niveau de la connaissance mais plutôt au niveau de l'action : cette illusion, entretenue par le « sens imaginaire de la loi » (causalité), masque le fait que l'autorité est sans vérité – que l'on obéit à la loi. dans la mesure où c'est incompréhensible. N'est-ce pas un calcul ? Par l'illusion du sens et de la causalité, ne dissimulons-nous pas le fait que nous ne comprenons pas et ne pouvons pas pleinement comprendre ni contrôler le calcul ? Que les ordinateurs se conçoivent de plus en plus les uns les autres et que notre utilisation est, dans une certaine mesure, une supplication, une foi aveugle ? La nouvelle rhétorique de « l'interactivité » obscurcit plus qu'elle ne révèle.

Les systèmes d'exploitation créent également des utilisateurs de manière plus littérale, car les utilisateurs sont une construction de système d'exploitation. Les connexions utilisateur sont apparues avec les systèmes d'exploitation en temps partagé, comme UNIX, qui encouragent les utilisateurs à croire que les machines sur lesquelles ils travaillent sont leurs propres machines (avant cela, les ordinateurs utilisaient principalement le traitement par lots ; avant cela, on faisait réellement fonctionner l'ordinateur, donc il n'y avait pas besoin de systèmes d'exploitation - il y avait des opérateurs humains). Comme l'ont soutenu de nombreux historiens, les systèmes d'exploitation à temps partagé développés dans les années 1970 ont engendré « l'ordinateur personnel ».57

Le logiciel et l'idéologie s'accordent parfaitement car tous deux tentent de cartographier les effets matériels de l'immatériel et de positionner l'immatériel à travers des indices visibles. Grâce à ce processus, l'immatériel émerge comme une marchandise, comme quelque chose à part entière. Ainsi, la description par Broy des pionniers comme cherchant à rendre le logiciel plus facile à visualiser ressemble étrangement au logiciel lui-même, car qu'est-ce qu'un logiciel sinon l'effort même de rendre quelque chose explicite, de rendre visible quelque chose d'intangible, tout en rendant en même temps le visible (comme comme la machine) invisible ? Bien que le parallèle entre logiciel et idéologie soit convaincant, il est important de ne pas s'arrêter ici, car réduire l'idéologie au logiciel vide l'idéologie de sa critique du pouvoir – quelque chose d'absolument essentiel à toute théorie de l'idéologie58 . structure, agit à la fois comme idéologie et comme critique de l'idéologie – comme moyen de dissimulation et de révélation, brise également l'analogie entre logiciel et idéologie. La puissance du logiciel réside dans cette double action et dans le visible qu'il rend invisible, un effet du fait que les langages de programmation deviennent une tâche linguistique.

Voir à travers la transparence Lorsque vous
dessinez un lapin d'un chapeau, c'est parce que vous l'avez mis là en premier lieu.

—Jacques Lacan59

Cet acte de révélation anime les bases de données et autres structures essentielles à la « transparence » ou à ce que Baudrillard appelait « l'obscénité » de la communauté.

communication. Bien que l'imagerie numérique joue certainement un rôle dans la notion de transparence des réseaux informatiques, elle n'est pas la seule, ni la clé. Prenons, par exemple, « The Matrix », un programme multiétatique qui passe au crible des bases de données d'informations publiques et privées, apparemment pour trouver des criminels ou des terroristes. Ce programme fonctionne en intégrant

> informations provenant de sources disparates, telles que les immatriculations de véhicules, les données sur les permis de conduire, les antécédents criminels et les dossiers immobiliers, et les analyser pour détecter des schémas d'activité susceptibles de faciliter les enquêtes des forces de l'ordre. Le matériel promotionnel de l'entreprise affirmait : « Lorsqu'un nombre suffisant de données apparemment insignifiantes sont analysées par rapport à des milliards d'éléments de données, l'invisible devient visible. »60

Même si ses partisans prétendent que « la Matrice » rassemble simplement des informations déjà disponibles pour les forces de l'ordre,

> Les opposants au programme affirment que la capacité des réseaux informatiques à combiner et à filtrer des montagnes de données amplifie considérablement le pouvoir de surveillance de la police, exposant ainsi des personnes innocentes à un risque accru d'être empêtrées dans des rafles de données. Le problème est aggravé, disent-ils, dans un monde où de nombreux aspects de la vie quotidienne laissent des traces en ligne.61

Le 15 mars 2004, plus des deux tiers des États ont retiré leur soutien à « Matrix », invoquant des problèmes de budget et de confidentialité. « La Matrice » était considérée comme une violation de la vie privée parce qu'elle rendait visible l'invisible (encore une fois, l'action du logiciel lui-même), et non parce que l'ordinateur reproduisait des images indexicales. Cela a amplifié le pouvoir de la police en lui permettant d'établir des liens faciles. La Total Information Agency, un projet du gouvernement américain visant à rassembler ses différentes bases de données électroniques, a été de la même manière décriée et pratiquement supprimée par le Congrès américain en 2003.

À un niveau plus personnel, l'informatique permet des connexions en rendant l'invisible visible et anime les interfaces informatiques personnelles. En tapant Word, des lettres apparaissent sur mon écran, représentant ce qui est stocké de manière invisible sur mon ordinateur. Ma saisie et mon clic semblent avoir des actions correspondantes sur l'écran. En ouvrant un fichier, je le rends visible. À tous les niveaux, les logiciels semblent donc avoir pour objectif de rendre visible l'invisible, de traduire entre un code lisible par ordinateur et un langage lisible par l'homme. Manovich s'empare de cette traduction et fait du « transcodage » – la traduction de fichiers d'un format à un autre, qu'il extrapole aux relations entre couches culturelles et informatiques – son cinquième et dernier principe des nouveaux médias dans The Language of New Media. Manovich soutient que pour comprendre les nouveaux médias, nous devons impliquer les deux couches, car même si la couche superficielle peut ressembler à tous les autres médias, la couche cachée

C'est au niveau du calcul que réside la véritable différence entre les nouveaux et les anciens médias – la programmabilité. Il soutient ainsi que nous devons passer des études des médias aux études sur les logiciels, et que le principe du transcodage est une façon de commencer à réfléchir aux

études sur les logiciels.62 Le problème avec la notion de transcodage de Manovich est qu'elle se concentre sur les données statiques et traite le calcul comme un simple traduction. La programmabilité ne signifie pas seulement que les images peuvent être manipulées de nouvelles manières, mais également que l'ordinateur agit constamment d'une manière indépendante de notre volonté. Considérer le logiciel comme un simple « transcodage » efface le calcul nécessaire au fonctionnement des ordinateurs. La lecture trompeuse de l'ordinateur ne se contente pas de traduire ou de transcoder le code en texte/image/son ou vice versa ; sa lecture – qui confond lecture et écriture (pour un ordinateur, lire, c'est écrire ailleurs) – participe aussi à d'autres lectures invisibles. Par exemple, lorsque le Media Player de Microsoft lit un CD, il envoie à Microsoft Corporation des informations sur ce CD. Lorsqu'il lit un fichier Real Media, tel qu'un clip vidéo de CNN, il envoie à CNN son « identifiant unique ». Vous pouvez choisir de travailler hors ligne lors de la lecture d'un CD et demander à votre lecteur multimédia de ne pas transmettre son « identifiant unique » lorsqu'il est en ligne, mais ces choix nécessitent deux modifications des paramètres par défaut. En installant Media Player, vous avez également accepté d'autoriser Microsoft à « fournir des mises à jour de sécurité pour les composants du système d'exploitation qui seront automatiquement téléchargées sur votre ordinateur. Ces mises à jour liées à la sécurité peuvent désactiver votre capacité à copier et/ou lire du contenu sécurisé et à utiliser d'autres logiciels sur votre ordinateur. »63 Fondamentalement, Microsoft peut modifier les composants de votre système d'exploitation sans préavis ni votre consentement explicite. Ainsi, pour créer un ordinateur plus « sécurisé », où « sécurisé » signifie protégé de l'utilisateur, Microsoft peut désactiver les fichiers et applications piratés et/ou signaler leur présence à sa base de données principale.64 Bien entendu, les publicités de Microsoft ne mettent pas l'accent sur les mécanismes de suivi du Media Player, mais vendez-le plutôt comme étant responsabilisant et convivial. Vous pouvez désormais écouter vos stations de radio sur CD et sur Internet en un seul clic de souris : c'est comme votre boombox, mais en mieux. Vous pouvez désormais recevoir automatiquement les mises à jour logicielles et optimiser votre connexion aux sites distants.

Soyons clairs : cet article n'est pas un appel à un retour à une époque où l'on pouvait voir ce que l'on fait. Ces jours sont révolus depuis longtemps. Comme le soutient Kittler, fondamentalement, nous n'écrivons plus – grâce à notre utilisation de traitements de texte, nous avons confié cette tâche aux ordinateurs.65 Il ne s'agit pas non plus d'une accusation contre les logiciels ou la programmation (moi aussi je suis influencé et amoureux du plaisir causal des logiciels).). Il s'agit cependant d'un argument contre les notions de logiciel de sens commun, précisément en raison de leur statut de sens commun (et en ce sens, elles répondent à la notion Gramscienne de l'idéologie en tant que sens commun hégémonique) ; à cause des histoires et des regards qu'ils effacent ; et à cause de l'avenir, ils

pointer vers. Le logiciel est devenu un raccourci de bon sens pour
la culture et le matériel sont un raccourci pour la nature. (Dans le débat actuel
Dans le cadre de la recherche sur les cellules souches, les cellules souches ont été qualifiées de « matériel ».
Historiquement, les logiciels facilitaient également la séparation des motifs
matière, nécessaire à la séparation des gènes de l'ADN.66) Dans notre société
dite postidéologique, le logiciel entretient et dépolitise
notions d'idéologie et de critique de l'idéologie. Les gens peuvent nier l'idéologie,
mais ils ne nient pas le logiciel – et ils lui attribuent, métaphoriquement, des
pouvoirs plus grands que ceux qui ont été attribués à l'idéologie. Notre
les interactions avec les logiciels nous ont disciplinés, ont créé certaines attentes
concernant les causes et les effets, nous ont offert du plaisir et du pouvoir que nous
je pense qu'il devrait être transférable ailleurs. La notion de logiciel a
se sont glissés dans notre vocabulaire critique de manière pour la plupart sans interrogation.67
En interrogeant le logiciel et les connaissances visuelles qu'il perpétue, nous
peut dépasser la soi-disant crise de l'indexicalité pour comprendre les nouvelles
façons dont la connaissance visuelle est transformée et perpétuée, et non
simplement déplacée ou rendue obsolète.

Remarques

1. Cité par John Schwartz, « Privacy Fears Erode Support for a Network to Fight Crime », The New York Times, 15 mars 2004, C1. Seisint est une société développant « la Matrice », un système informatique évoqué plus loin dans cet article.

2. Jean Baudrillard, L'Extase de la communication, trad. Bernard Schutze et Caroline Schutze (Brooklyn : Semiotext(e), 1988), 21-22 ; accent dans l'original.

3. Qu'est-ce que Microsoft Word, par exemple ? S'agit-il de l'exécutable chiffré, résidant sur un CD ou sur son disque dur (ou même de son code source), ou de son exécution ? Surtout, les deux ne sont pas identiques même s'ils s'appellent tous deux Word : non seulement ils se trouvent à des endroits différents, mais l'un est un programme tandis que l'autre est un processus. La programmation structurée , comme nous le verrons plus loin, a joué un rôle clé dans l'amalgame entre programme et processus.

4. Voir Lev Manovich, The Language of New Media (Cambridge : MIT Press, 2001), p. 48.

5. Paul Ceruzzi, Une histoire de l'informatique moderne, 2e éd. (Cambridge : MIT Press, 2003), 80.

6. Manfred Broy, « Software Engineering—From Auxiliary to Key Technology », dans Software Pioneers : Contributions to Software Engineering, éd. Manfred Broy et Ernst Denert (Berlin : Springer, 2002), 11-12.

7. Friedrich Kittler, « There Is No Software », ctheory.net, 18 octobre 1995, http://www.ctheory.net/text_file.asp?pick=74.

8. Dans les années 1950, le terme logiciel était rarement utilisé et faisait référence à tout ce qui complétait le matériel, comme la configuration des fiches des câbles. Dans les années 1960, le logiciel est devenu plus étroitement compris comme ce que nous appellerions aujourd'hui un logiciel système et plus largement comme « toute combinaison d'outils, d'applications et de services achetés auprès d'un fournisseur extérieur ». Thomas Haigh, « Les logiciels d'application dans les années 1960 en tant que concept, produit et service », IEEE Annals of the History of Computing 24, no. 1 (janvier-mars 2002) : 6. Pour en savoir plus à ce sujet, voir « The Style of Source Codes » de Wolfgang Hagen, trans. Peter Krapp dans Nouveaux médias, anciens médias : un lecteur d'histoire et de théorie, éd. Wendy Hui Kyong Chun et Thomas Keenan (New York : Routledge, 2005).

9. Voir Herman Goldstine et Adele Goldstine, « The Electronic Numerical Integrator and Computer (ENIAC) » IEEE Annals of the History of Computing 18 no. 1 (printemps 1996) : 10-16. Les deux activités ont été divisées par Arthur Burks en « programmation numérique » (mise en œuvre par les opérateurs) et « programmation proprement dite » (conçue par les ingénieurs et mise en œuvre par l'unité de programmation principale). Arthur Burks, « De l'ENIAC à l'ordinateur à programme stocké : deux révolutions dans les ordinateurs », dans Une histoire de l'informatique au vingtième siècle, éd. N. Metropolis et al., 311-344 (New York : Academic Press, Inc. et Harcourt Brace Jovanovich, 1980). La programmation numérique traitait des spécificités du problème et des unités arithmétiques et était similaire à la microprogrammation ; la programmation proprement dite traitait des unités de contrôle et de la séquence des opérations.

10. Voir Martin Campbell-Kelly et William Aspray, Computer : A History of the Information Machine (New York : Basic Books, 1996), 184.

11. Voir Wolfgang Hagen, « The Style of Source Codes », dans New Media, Old Media, éd. Wendy Hui Kyong Chun et Thomas Keenan (New York : Routledge, 2005).

12. Adele Mildred Koss, « Programmation sur l'Univac 1 : le récit d'une femme », Annales IEEE de l'histoire de l'informatique 25, non. 1 (janvier-mars 2003) : 56.

13. Voir Jacques Derrida, « Signature Event Context », dans Jacques Derrida, Limited Inc., trans. Samuel Weber et Jeffrey Mehlman, 1-23 (Evanston : Northwestern University Press, 1988).

14. John Backus, « La programmation en Amérique dans les années 1950 – Quelques impressions personnelles », Une histoire de l'informatique au vingtième siècle, éd. Métropole et

al., 127.

15. Koss, 58 ans.

16. Jean Sammett, Langages de programmation : histoire et principes fondamentaux (Englewood Cliffs : Prentice-Hall, 1969), 144.

17. Velours, 148.

18. Voir par exemple Theodor Nelson, Computer Lib (Richmond, WA : Tempus Books of Microsoft Press, 1987).

19. Harry Reed, « Ma vie avec l'ENIAC : A Worm's Eye View », dans Cinquante ans d'informatique militaire, de l'ENIAC au MSRC, Army Research Laboratory, 2000, 158 ; accessible en ligne sur : http://ftp.arl.mil/~mike/comphist/harry_reed.pdf.

20. Voir « Anecdotes : Comment êtes-vous arrivé à l'informatique ? » Annales IEEE de la Histoire de l'informatique 25, non. 4 (octobre-décembre 2003) : 48-59.

21. Paul N. Edwards, The Closed World : Computers and the Politics of Discourse in Cold War America (Cambridge : MIT Press, 1996), p.

22. IJ Bien. « Travail pionnier sur les ordinateurs à Bletchley », dans A History of L'informatique au XXe siècle, éd. Metropolis et coll., 31-46.

23. Cité dans Ceruzzi, 82.

24. Voir Michael S. Mahoney, « Trouver une histoire pour le génie logiciel », IEEE Annales de l'histoire de l'informatique 27, non. 1 (janvier-mars 2004) : 8-19.

25. Comme indiqué précédemment, bien que le logiciel produise des effets visibles, le logiciel lui-même ne peut pas être vu. Luce Irigaray, dans This Sex Which Is Not One (trad. Catherine Porter, Ithaca, NY : Cornell UP, 1985), a également soutenu que la sexualité féminine est non visuelle, « son organe sexuel représente l'horreur de rien à voir » ; souligné dans l'original, 26.

26. Sadie Plant, Zeros + Ones: Digital Women + The New Technoculture (Nouveau York : Doubleday, 1997), p. 37.

27. Campbell-Kelly et Aspray, 181.

28. Koss, 51 ans.

29. Koss, 49 ans.

30. Fitz, 21 ans.

31. Paul Edwards, « L'armée et le micromonde : les ordinateurs et la politique de l'identité de genre », Signs 16, no. 1 (1990) : 105, 125 ; italiques ajoutés.

32. Vannevar Bush décrit ainsi l'action d'un sténographe : « Une jeune fille caresse ses touches avec langueur et regarde la pièce et parfois l'orateur avec un regard inquiétant. » Vannevar Bush, « As We May Think », The Atlantic Monthly, juillet 1945, repr. 1994, http://www.ps.uni-sb.de/~duchier/pub/vbush/vbush.txt. Les femmes ont été largement déplacées des programmeurs vers les utilisateurs, tout en continuant à dominer le bassin de main-d'œuvre de la production de matériel informatique, même si ces emplois se sont déplacés à travers le Pacifique.

33. Pour en savoir plus sur la crise du logiciel et ses relations avec le génie logiciel, voir Cambell-Kelly et William Aspray, 196-203, Ceruzzi, 105, et The Mythical Man-Month: Essays on Software Engineering, 20e de Frederick P. Brooks. Édition anniversaire (NY : Addison-Wesley Professional, 1995).

34. Mahoney, 15 ans.

35. Edsger W. Dijkstra, « EWD 1308 : What Led to 'Notes on Structure Programming' », dans Software Pioneers, éd. Broy et Denert, 342.

36. Edsger W. Dijkstra, « Aller à la déclaration considérée comme nuisible », dans Software Pionniers, éd. Broy et Denert, 352, 354.

37. On peut soutenir que les visites sont nuisibles car ce sont des moments dans lesquels l'ordinateur est directement adressé – les branchements conditionnels et les boucles permettent à un programme de passer à une autre section sans une telle adresse. Cette adresse pose la question :

exactement qui ou quoi doit aller dans une autre section ? Curieusement, go to est une traduction de la commande d'assemblage « transfer control to » – pour donner le contrôle du programme à la commande située à l'adresse X. La différence entre go to et transfer control to est à l'origine de la crise d'agence au cœur de la programmation.

38. John V. Guttag, « Types de données abstraites, hier et aujourd'hui », dans Software Pioneers, éd. Broy et Denert, 444.

39. Guttag, 444.

40. Guttag, 445.

41. David Eck, La machine la plus complexe : une enquête sur les ordinateurs et l'informatique (Natick, MA : AK Peters, 2000), 329, 238.

42. Koss, « Programmation sur l'Univac 1 : le récit d'une femme », 49.

43. Paul Edwards, « L'armée et le micromonde », p. 108-9.

44. Pour en savoir plus sur le plaisir de programmer, voir Linus Torvalds, Just for Fun : The Story of an Accidental Revolutionary (New York : HarperBusiness, 2001) ; et Eben Moglen, « L'anarchisme triomphant : le logiciel libre et la mort du droit d'auteur », Premier lundi 4, non. 8 (2 août 1999), http://firstmonday.org/issues/issue4_8/moglen/index.html .

45. Voir la discussion d'Edwards sur le travail de John McCarthy dans The Closed World, p. 258.

46. John von Neumann, Articles de John von Neumann sur l'informatique et la théorie informatique, éd. William Aspray et Arthur Burks (Cambridge : MIT Press, 1987), 413.

47. Ben Schneiderman, « Manipulation directe : un pas au-delà des langages de programmation », dans The New Media Reader, éd. Noah Wardrip-Fruin et Nick Montfort (Cambridge : MIT Press, 2003), 486.

48. Brenda Laurel, Computers as Theatre (Reading, MA : Addison-Wesley Publishers, 1991), xviii.

49. Lev Manovich, « Generation Flash », 2002, http://www.manovich.net/DOCS/generation_flash.doc .

50. Cette notion d'autonomisation soulève la question de savoir ce que signifie produire du « code original » et comment la conception du Bauhaus au service de la conception de l'information n'est pas une citation. Manovich, « Génération Flash ».

51. Manovitch, « Génération Flash ».

52. Manovitch, « Génération Flash ».

53. Fredric Jameson, Le postmodernisme ou la logique culturelle du capitalisme tardif (Durham : Duke University Press, 1991), p. 51.

54. Voir Wendy Hui Kyong Chun, Contrôle et liberté : pouvoir et paranoïa dans l'ère de la fibre optique (Cambridge : MIT Press, 2005).

55. Louis Althusser, « Idéologie et appareils idéologiques d'État (Notes vers une enquête) », dans Lénine et Philosophie et autres essais, trad. Ben Brewster (New York : Monthly Review Press, 2001), 109.

56. Voir Slavoj Zizek, The Sublime Object of Ideology (Londres : Verso, 1989), 11-53.

57. Voir Ceruzzi, 208-209 ; Campbell-Kelly, 207-29.

58. Cependant, ces parallèles révèlent sans doute le fait que notre compréhension de l'idéologie fait défaut précisément dans la mesure où elle repose, comme les interfaces, sur un modèle de comportement fondamentalement théâtral.

59. Jacques Lacan, Le Séminaire de Jacques Lacan, Livre II : Le Moi dans la théorie de Freud et dans la technique de la psychanalyse (1954-1955), éd. Jacques-Alain Miller, trad. Sylvana Tomaselli (New York : Norton, 1991), 81.

60. Schwartz, C1.

61. Schwartz, C1.

62. Manovitch, Le langage des nouveaux médias, p. 48.

63. Voir « Contrat de licence » de Media Player.

64. Microsoft envisage de telles actions dans son initiative Palladium. Voir Florence Olsen, « Problèmes de contrôle : le plan de Microsoft visant à améliorer la sécurité informatique pourrait déclencher une lutte pour l'utilisation des documents en ligne », Chronicle of Higher Education, 21 février 2003, http://chronicle.com/free/v49/i24/24a02701. htm.

65. Voir Kittler, « Il n'y a pas de logiciel ».

66. Voir François Jacob, La logique de la vie : une histoire de l'hérédité, trad. Betty E. Spillman (New York : Pantheon Books, 1973), 247-98.

67. Le concept de « logiciel rhétorique » de Richard Doyle, développé dans son ouvrage On Beyond Living : Rhetorical Transformations of the Life Sciences (Stanford : Stanford University Press, 1997), incarne l'utilisation du logiciel comme terme critique dans le discours scientifique non scientifique et révèle à quel point le logiciel structure nos idées et se présente donc comme le concept qui nécessite le plus d'interrogation.