SUR LES ORIGINES DU LOGICIEL

Paolo Rocchi
IBM, via Shangai 53, Roma, ITALY
paolorocchi@it.ibm.com

Abstract: The ever-increasing diffusion of software products is pushing researchers to clarify the special nature of programming. Several authors tackle the problem from a narrow perspective that although appears insufficient to justify this original engineering. We present a logical framework and infer the general features of programming from this theoretical base; notably this method justifies the program maintenance and the software structures, given as facts so far.
The key concepts of this study, illustrated in professional courses, proved to mature the student minds. In particular, the lessons have made the attendees, who are technicians and managers, better competent in software updating that is one of the most acute problems in the field.

Résumé : La diffusion des produits logiciels sur le marché mondial a conduit les chercheurs à éclaircir la nature spécifique de ces produits. Toutefois, certains ont abordé la question sous un angle restreint, en avançant des justifications jugées insuffisantes tant au plan culturel que scientifique. Le présent travail s'efforce de dépasser ces limites en exposant un contexte plus ample, afin d'illustrer les caractéristiques de l'ingénierie logicielle en suivant un processus logique. En particulier, nous avons retenu une approche rationnelle pour justifier l'introduction de la programmation, de la maintenance et des structures des programmes, à propos desquels on nous propose aujourd'hui des arguments plutôt dogmatiques.
Les concepts clé présentés ici ont été illustrés dans le cadre d'une formation professionnelle, qui a permis aux techniciens et aux dirigeants fréquentant les cours d'affiner leur appréhension du sujet, notamment en les aidant à gérer l'aspect *mise à niveau du logiciel*, qui reste dans la pratique l'une des questions les plus problématiques à résoudre.

1 INTRODUCTION

La diffusion croissante des applications logicielles sur le marché mondial nous amène à éclaircir leur nature spécifique en termes conceptuels. De nombreux articles abordent déjà la question, mais ils le font en ne se focalisant que sur des thèmes particuliers ou en considérant les choses sous un angle beaucoup trop restreint. Citons par exemple les travaux traitant des spécificités du logiciel en rapport à des projets complexes [Reifer 1993], à la formation technique [Ellis et al. 2002], à la métrique [Nikora et al. 2003], à la protection légale des droits d'auteur [Johnson-Laird 1992]. D'autres études, qui se proposent également de mettre en lumière des aspects plus généraux, finissent par ne plus analyser que les aspects pratiques et opérationnels du logiciel [Kruchten 2004][Boehm 2000]. En dernière approche, nous pourrions dire que les auteurs examinent la technologie en elle-même et par elle-même, en évitant d'approfondir les raisons qui en sont à l'origine et qui permettraient de l'éclaircir de manière définitive. En d'autres termes, ils font l'impasse sur l'étude du contexte qui génère la programmation et en éclaircit la nature, les possibilités et les limites. La compréhension exhaustive du logiciel a pris du retard, ce qui signifie que nous sommes dans l'incapacité de définir les questions techniques sur la base d'axiomes généraux. Il nous manque une théorie qui fournirait un apport réel aux opérations et aux décisions inhérentes au secteur informatique, raison pour laquelle les techniciens ne peuvent évaluer ni déduire des solutions dans un cadre rationnel, comme c'est par exemple le cas en physique. Cette lacune conduit à sous-estimer certains problèmes tels que la maintenance

logicielle, qui reste dans la pratique l'un des problèmes à résoudre les plus délicats et répandus [Glass 1998].
La présente recherche entend donc contribuer à éclaircir ces questions théoriques et pratiques, en proposant un contexte systémique et en discutant de la programmation suivant un processus strictement logique. En particulier, nous donnons les définitions de quatre éléments dont nous illustrons les propriétés; ensuite nous déduisons six propositions qui justifient autant d'aspects du logiciel.

2 LE CONTEXTE

La contextualisation du logiciel commence par le concept d'information.

2.1 INTRODUCTION À L'INFORMATION

Depuis longtemps que le débat est ouvert sur la nature de l'information, la conclusion ne semble pas encore à portée de main, au contraire. Toutefois les positions des chercheurs convergent au moins sur les premières étapes de l'analyse, en étant d'accord sur les points suivants :

2.1.A Définition: L'information est une entité physique et distincte.
2.1.B Définition: L'information représente un objet, un événement, etc.

Ferdinand de Saussure fut le premier à préciser les concepts de *signifiant* (ou *signifier*) pour dénoter le corps du signe. [Saussure 1974]. Les sémioticiens et les linguistes utilisent quantité de termes tels que '*expression, representamen, forme, mot, significant, token, structure superficielle, information carrier*' qui, tous, indiquent substantiellement un même concept. Dans le cadre du présent travail, le terme d'*information* entend indiquer une entité physique bien distincte symbolisant quelque chose, pour tenter ainsi de simplifier une telle abondance de terminologie. Le point 2.1.A définit l'aspect physique de l'information, sur lequel il ne fait doute; la plupart des chercheurs, y compris les symbolistes, l'acceptent d'un commun accord. Le point 2.1.B introduit la sémantique, qui constitue le point nodal de la recherche actuelle, mais que nous pouvons éviter d'approfondir ici. Les définitions 2.1.A et 2.1.B suffisent aux finalités du présent article, comme nous le verrons mieux par la suite. Elles sont en outre largement généralistes, soit parce qu'elles ne se réfèrent à aucun langage en particulier, soit parce qu'elles concernent tous les objets et les événements physiques présents dans la nature et constituant autant de formes potentielles de signal.

2.2 LES SYSTÈMES OPÉRATIONNELS

Le présent travail pose comme postulat que le *système général* (SG) a une nature opérationnelle, c'est-à-dire qu'il est dédié aux opérations d'acquisition, de transformation et de transport des produits. Les points 2.1.A et 2.1.B ont le mérite de consolider ce point de vue opérationnel et nous permettent de mettre le *système informationnel* (SI) sur un plan d'égalité avec tout autre système, puisque, de fait, SI manipule des informations qui sont des éléments physiques. C'est bien un ensemble de fonctions réalisées par des unités technologiques, biologiques, personnelles, sociales [Banville et al. 1989]. Nous pouvons citer comme exemples les bibliothèques, les bureaux, les services et les administrations, les appareils numériques et analogiques, le système nerveux de l'homme et des animaux. Une très vaste classe qui traite des signaux électriques,

chimiques, mécaniques, etc., ce qui corrobore le postulat 2.1.A. Nous posons donc la définition suivante :

2.2.A Définition: Le système général traite des entités matérielles, alors que le système informationnel est plus particulièrement spécialisé dans la gestion des informations.

Une telle approche opérationnelle est commune dans la littérature. Certains auteurs identifient même les systèmes informationnels avec les machines et les appareils. Par exemple, pour illustrer les qualités des SI, Alter emploie explicitement les termes suivants : « An information system is a system that uses information technology to c*apture, transmit, store, retrieve, manipulate, or display* information used in one or more business processes (Un système informationnel est un système qui utilise les technologies de l'information pour *capturer, transmettre, stocker, récupérer, manipuler ou visualiser* les informations utilisées dans/par un ou plusieurs processus métier.)» [Alter 1996].

2.3 PRINCIPALES PROPRIÉTÉS DU SYSTÈME INFORMATIONNEL

Le point 2.1 implique que le système informationnel est semblable à tout autre système, si ce n'est le fait que le produit traité se caractérise par les propriétés visées aux points 2.1.A et 2.1.B. Ce qui veut dire que les opérations d'un système informationnel sont capables de distinguer les signaux, alors qu'un système général n'est pas capable de percevoir les différences entre un élément et un autre.

2.3.A Propriété: Le système informationnel est capable de distinguer des entités physiques, là où un système général quelconque est incapable de les distinguer.

Le système général est gravement limité lorsqu'il doit faire face à des situations changeantes. En effet, le système général opère habituellement au sein du système-univers (SU) : le marché ou l'habitat, l'organisation politique internationale ou la collectivité locale. L'expérience enseigne que les actions de SU, qui sont capables d'influencer de manière déterminante le système général, ne suivent typiquement aucune règle.

2.3.B Définition: Le système-univers influe de façon casuelle sur le système général.

Le système général se heurte à des obstacles importants lorsque le système-univers opère des changements, puisqu'il est incapable de les percevoir. Il pourrait modifier ses actions et s'adapter à l'environnement, mais l'impossibilité de cette perception empêche une telle issue. Un handicap aggravé lorsque SU évolue de façon favorable, puisque le système général est alors dans l'impossibilité de tirer profit de ces opportunités positives. La propriété 2.3.A nous suggère donc d'exploiter SI pour suppléer aux limites de SG. Dans la réalité, le système informationnel s'ajoute au système général pour remédier à l'incapacité qu'a celui-ci d'y voir clair et de décider. Nous pourrions dire que le système général est aveugle, et que le système informationnel lui permet de surmonter les difficultés imprévues.

2.3.C Propriété: Le système informationnel est capable d'améliorer la conduite du système général pour assurer la survie de ce dernier.

Formellement, nous pouvons définir l'évolution structurelle visée en 2.3.C de la façon suivante :

$$SG \; \eta \; (SG + SI) \qquad (1)$$

Une très vaste bibliographie confirme que le système informationnel aide l'entreprise à survive de la meilleure façon possible [Aguilar 1967]. D'autres systèmes informationnels poursuivent le même objectif, en particulier le cerveau humain, qui jouit aussi de la propriété 2.3.C. Je me limiterai ici à rappeler Herbert Spencer, qui a élargi les idées de Darwin aux processus informationnels, lorsqu'il illustre la capacité évolutive de l'espèce générée par l'utilisation du cerveau [Young 1970]. Plus récemment encore, ce concept a été repris par les analyses sur la survie de l'homme dans le monde moderne [Schmitt 2002].
Le point 2.3.C implique que le système informationnel n'est pas un simple ajout fait au système général, mais plutôt que SI amène SG à la réussite, c'est-à-dire que le système informationnel envoie des messages au système général en lui imposant de parcourir des voies précises ou de se comporter de telle ou telle façon.

2.3.D Propriété: Le système informationnel a un rôle directif.

Nous pouvons formaliser le point 2.3.D par différents niveaux hiérarchiques [Whyte et al. 1969]. Écrivons donc la formule structurelle ci-après, où la deuxième partie de l'équation indique que SI se trouve à un niveau hiérarchique supérieur à celui du système général :

$$(SG + SI) = \frac{SI}{SG} \qquad (2)$$

La hiérarchie ést la propriété constitutive des opérations informationnelles, nous élargirons ce concept en concluant que le système informationnel s'organise sur plusieurs niveaux [Harris 1998].

$$\begin{array}{ll} SI_{I} & \textit{niveau 1} \\ \hline SI_{II} & \textit{niveau 2} \\ \hline SI_{III} & \textit{niveau 3} \end{array} \qquad (3)$$

Le point 2.3.B affirme que les changements de SU – qui ont soit des effets positifs soit négatifs sur le système général – sont imprévisibles, et que par conséquent personne ne peut anticiper afin d'établir le meilleur comportement à tenir pour le système informationnel. La stratégie nécessaire pour faire face au système-univers ne peut donc être fixée à l'avance. Et même si les obstacles ont tous été surmontés de façon positive dans le passé, aujourd'hui le système informationnel doit inventer ses propres contre-mesures pour faire face aux nouvelles évolutions de SU. Concluons en énonçant ci-après une propriété générale du système informationnel :

2.3.E Propriété: Les activités du système informationnel sont imprédictibles.

La proposition 2.3.E trouve de nombreuses confirmations dans la littérature. L'inventivité d'un système informationnel aussi spécial que celui du cerveau humain est largement traitée [Chalmers 2002] ; l'Intelligence Artificielle a aussi ouvert le débat sur les capacités d'invention des machines [Mitchell 1997].

Nous pourrions objecter que de très nombreux systèmes informationnels travaillent selon des procédures précises et ne sont absolument pas inventifs. Par exemple, Monsieur SG, malade, appelle son médecin, SIm, qui établit le diagnostic de la maladie et suit un protocole précis pour le soigner.
L'explication est la suivante : le système IS, qui opère en suivant des règles, est précédé et étayé par un système beaucoup plus vaste, qui a trouvé la procédure à suivre après nombre d'essais et de tentatives, une procédure qui n'était pas connue à l'avance. Par exemple, le médecin SIm est guidé par la recherche médicale SIr, qui a découvert la thérapie. SIr est un système énorme, qui travaille durant des siècles pour rechercher la meilleure solution. Par conséquent le système dans son ensemble (SIm + SIr) est globalement imprédictible et confirme la proposition 2.3.E. On retrouve partout le couple que composent le système inventif qui a trouvé la procédure et le système qui l'applique : dans le milieu économique, dans les affaires, l'ingénierie, la recherche scientifique, la littérature et dans de nombreux autres secteurs.

3 LES RAISONS DU LOGICIEL

Voyons maintenant de quelle manière le contexte systémique précédemment illustré exerce une influence directe sur la construction des calculateurs, et comment il permet d'éclaircir les raisons qui sont à la base de la programmation.

3.1 LES ORIGINES DU LOGICIEL ET DE LA MAINTAINANCE DES PROGRAMMES

Il est tout à fait notoire que l'ordinateur est un système informationnel [Alter 1996]. Pour autant, il se caractérise par la propriété visée au point 2.3.A et vient au secours de SG. Il doit faire face aux changements imprévus provoqués par le système-univers, en vertu de quoi son comportement n'est pas totalement déterminable à l'avance.

3.1.A Proposition: Les opérations de l'ordinateur sont imprédictibles.

Cette caractéristique nous permet de déduire qu'au sein même de l'usine où la machine devrait être pensée et réalisée dans toute sa complétude, nul n'est en mesure d'anticiper dans le détail les opérations que la machine exécutera par la suite. Cette contrainte sévère impose donc des modifications méthodologiques, et suggère en particulier de séparer en deux étapes la construction d'un ordinateur. Posons que la première étape aura lieu à l'usine, où l'opération ne pourra cependant pas être menée à terme, et ajoutons que d'autres intervenants devront parachever la réalisation in situ, c'est-à-dire là où le système informationnel devra venir au secours du système général, là où des tâches précises pourront être confiées à l'ordinateur. En résumé, la construction de la machine devra être confiée à deux filières d'ingénierie distinctes, qui auront les caractéristiques suivantes :

3.1.B Proposition: L'ingénierie *initiale* crée un ordinateur incomplet.
3.1.C Proposition: L'ingénierie *finale* détermine les fonctions définitives de l'ordinateur.

Les présentes conclusions, qui découlent du contexte général visé au paragraphe 2, expliquent les origines de la technologie matérielle par le biais de la proposition 3.1.B*, et celles de la*

technologie logicielle par le biais de la proposition 3.1.C. Les méthodologies Hardware et Software, loin d'être antagonistes, coopèrent au contraire en vue d'une même fin.
La proposition 2.3.B nous dit que le comportement du système-univers se modifie de façon casuelle, et, partant, imprévue. Dès lors la finition de l'ordinateur doit être rapide, si l'on veut que le système informationnel puisse venir au secours du système général en temps utiles.

3.1.D Proposition: L'ingénierie du logiciel est une ingénierie souple.

Les informations étant les composantes physiques les plus agiles et les plus légères qui permettent de réduire les retards au minimum, pour répondre aux exigences de la proposition 3.1.D, l'ingénierie de finition visée au point 3.1.C doit faire appel aux informations. Le logiciel parachève la construction de l'ordinateur par l'écriture, qui reste la façon la plus rapide pour déterminer le comportement d'un système. Ce raisonnement établit que l'ordinateur sera nécessairement programmé par une séquence de symboles. Jusqu'à présent, le modèle de Turing n'a été proposé que de façon dogmatique ; or le cadre logique que nous traçons ici en fournit les raisons. Nous conclurons sur une note historique [Randell 1979]. Dans le passé, les techniciens ont tenté d'autres voies pour réaliser la finition des calculateurs, en utilisant par exemple les broches et les contacts électriques, qui se sont avérés être plus lents que la technique d'écriture, désormais universellement acceptée. En d'autres termes, nous parvenons aujourd'hui sur un plan rationnel aux mêmes conclusions qui ont pu être tirées hier, à force d'essais et de tentatives, sur un plan pratique.
Le système informationnel mécanique exécute des actions pour guider le système général en fonction des changements introduits par SU. D'ailleurs, il effectue d'autres opérations accessoires, subsidiaires ou autres. D'où il s'ensuit :

3.1.E Proposition: Les programmes logiciels se répartissent en deux catégories principales.

La première catégorie englobe les programmes qui répondent directement aux besoins imposés par SU; font partie de la deuxième catégorie tous ceux qui ont d'autres fonctions. D'habitude on les nomme '*programmes applicatifs*' et '*programmes de base*'. Les premiers sont souvent mis à jour dans l'urgence, compte tenu de leur rôle, puisque, de fait, les changements de SU se succédant sans aucune règle, ils arrivent à créer des contraintes stressantes pour les systèmes informationnels. Les programmes appartenant à la seconde catégorie, tels que les *systèmes d'exploitation*, les *programmes utilitaires*, les *programmes de calcul*, sont beaucoup plus stables. Toutefois, bien qu'ayant des caractéristiques opposées, ces deux types de programmes s'inscrivent dans le même cadre logique que nous avons tracé jusqu'à présent.

3.2 LES STRUCTURES LOGICIELLES

Le point 2.3.D implique que l'ordinateur a des propriétés hiérarchiques, et, plus particulièrement, les programmes qui modèlent de façon précise le système informationnel sont nécessairement de nature hiérarchique.

3.2.A Proposition: La hiérarchisation est la propriété fondamentale de la programmation.

Afin de corroborer cette conclusion, à laquelle nous sommes parvenus en suivant la voie théorique, nous commenterons ici cinq principaux composants logiciels pour démontrer la justesse de la proposition 3.2.A.

1° Les instructions du programme déterminent le fonctionnement des circuits matériels. Le plus élémentaire des composants logiciels a pour but de commander une action. La propriété hiérarchique démontre donc qu'elle en est la caractéristique de base, abstraction faite de toute fonction particulière requise par la commande.

2° Les instructions logicielles appartiennent à deux classes principales : les instructions de commande (comme, par exemple : *goto*, *jump*, *compare*), et les instructions de traitement (comme, par exemple : *multiply*, *xor*, *store*, *read*, *write*). Les premières déterminent l'exécution des secondes [Clements 2000], à savoir qu'elles se trouvent sur deux niveaux hiérarchiques distincts :

instructions de commande *niveau 1*

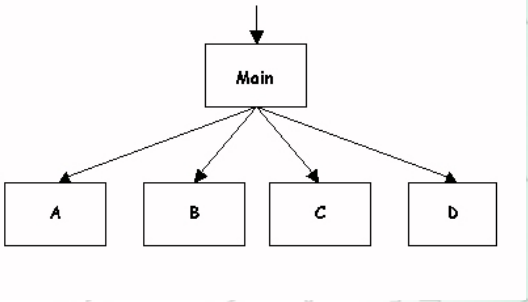
---

instructions externes/internes *niveau 2*

(4)

Cette propriété vaut également pour les commandes macros : *Do While*, *If Then*, *Call*, etc., qui conditionnent l'exécution des routines commandées.

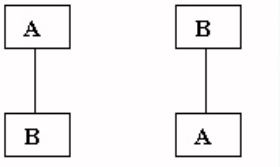
3° Lorsqu'il s'agit d'un programme de grandes dimensions, les techniciens le divisent en modules pour mieux le gérer. Les modules sont activés par la commande macro Call [Wyatt 2002], qui, par définition, établit une dépendance entre module appelant et module appelé.



(5)

Par conséquent, le 'software-package' comprend un module au sommet, appelé *Main***,** qui active les différentes fonctions. L'organigramme, ou *structure chart* [Wilcox 1990], donne une preuve évidente de la hiérarchie qui ordonne l'ensemble du « paquet ».

4° Dans la programmation par objets, supposons que la classe A ait besoin d'un service de la classe B, nous dirons alors que B dessert A, ce qui veut dire que la classe B se trouve à un niveau hiérarchique inférieur par rapport à celui de la classe A. La propriété hiérarchique est également essentielle dans un environnement à objets, avec cette particularité que les rapports peuvent s'inverser, lorsque l'on a par exemple A qui dessert B. [Emmerich 2000]



(6)

En conclusion, l'arbre hiérarchique établit la dépendance stricte et fixe entre les modules en (5), alors que la hiérarchie entre les classes varie en (6). Autant la configuration *maître-esclave* (5) que *client-serveur* (6) démontrent que la hiérarchie est la caractéristique essentielle de la programmation.

5° Le noyau du superviseur du système d'exploitation prend la forme (5) [Wulf et al. 1974]. Puisque le système d'exploitation organise l'ensemble des opérations, nous en concluons que l'organisation globale de l'ordinateur est hiérarchique.

Les auteurs n'ont pas l'habitude d'approfondir l'aspect hiérarchique qu'il y a dans la programmation. Ils abordent habituellement le sujet de façon indirecte, voir à titre d'exemple la Théorie de la Complexité [Ding et al. 2000] et le Graphe de Dépendance du Système (*System Dependence Graph*) [Ottenstein 1984], qui ne mentionnent la hiérarchie que comme un facteur subsidiaire. Cette superficialité n'aide manifestement pas à progresser dans la connaissance du logiciel.

4 CONCLUSIONS

Le présent travail suit une méthode purement rationnelle. En effet, il se compose d'une première partie qui propose un schéma systémique, suivi d'une deuxième partie qui en développe les conséquences afin d'éclaircir la nature du logiciel, le tout en utilisant de façon rigoureuse l'intellect. Nous avons ainsi évoqué la naissance même de la programmation et abordé des phénomènes aussi importants que la maintenance et la structuration.

Les principes exposés ici ont été enseignés dans le cadre de cours professionnels dispensés chez IBM, fortement appréciés par les techniciens et les dirigeants qui les ont fréquentés [Rocchi 2002]. En particulier, ils ont reconnu toute l'importance de la hiérarchie, que nous avons ramenée à un concept unique, là où elle est aujourd'hui disséminée sous diverses formes et associée à différentes solutions. En démontrant aux élèves que l'évolution des programmes applicatifs se caractérise par des mutations qui se produisent dans le temps non pas pour des raisons accidentelles, mais comme règle générale, nous leur avons permis de mûrir leur manière d'appréhender la maintenance, qui est encore à l'heure actuelle source de grosses difficultés.

Nous pensons enfin que la présente étude apporte une contribution à la recherche sur les systèmes généraux, puisqu'elle s'inscrit dans le cadre d'un travail plus vaste sur la question [Rocchi 2000].

RÉFÉRENCES

[Reifer 1993] Reifer D.J. (1993) - *Software Management* - IEEE Computer Soc. Press, Los Alamitos, Calif.

[Ellis et al. 2002] Ellis H.J.C., Mead N.R., Moreno A., Tanner C.D., Ramsey D. (2002) - Characteristics of Successful Collaborations to Produce Educated Software Engineering Professionals - *Computer Science Education*, 12(1-2), pp. 119-140.

[Nikora et al. 2003] Nikora A.P., Munson J.C. (Sept. 2003) - Understanding the Nature of Software Evolution - *Proc. Intl. Conference on Software Maintenance*, pp. 83-93.

[Johnson-Laird 1992] Johnson-Laird A. (1992) - Reverse Engineering of Software: Separating Legal Mythology from Actual Technology -*Software Legal J*., 5, pp.334-354.

[Kruchten 2004] Kruchten P. (April 2004) - Putting the "Engineering" into "Software Engineering" - *Proc. of the 2004 Australian Software Engineering Conference*, pp. 2-9.

[Boehm 2000] Boehm B., Basili V.R. (May 2000) - Gaining Intellectual Control of Software Development - *Computer*, 33(5), pp. 27-33.

[Glass 1998] Glass R. (July/August 1998)- Maintenance: Less is not More - *IEEE Software*, pp. 67-68.

[Saussure 1974] Saussure De F. (1974) - *Course in General Linguistics* - (trans. Wade Baskin) Fontana/Collins London.

[Banville et al, 1989] Banville C., Landry M. (1989) - Can the MIS Field be Disciplined ? - *Communications of the ACM*, 32(1).

[Alter 1996] Alter S. (1996) - *Information Systems: A Management Perspective* - Benjamin/Cummings.

[Aguilar 1967] Aguilar F.J. (1967) - *Scanning the Business Environment* - Macmillan, NY.

[Young 1970] Young R.M. (1970) - *Mind, Brain and Adaptation in the Nineteenth Century* - Oxford University Press.

[Schmitt 2002] Schmitt R. (February 2002) - Survive and Thrive Through Competitive Intelligence - *DM Direct Newsletter*.

[Whyte 1969] Whyte L.L., Wilson A.G., Wilson D. (eds.) (1969) - *Hierarchical structures* - American Elsevier, N.Y..

[Harris 1998] Harris D. (1998) - *Systems Analysis and Design for the Small Enterprise* - Thomson Learning, Toronto.

[Chalmers 2002] Chalmers D.J. (2002) - *Philosophy of Mind: Classical and Contemporary Readings* - Oxford University Press.

[Mitchell 1997] Mitchell T.M. (1997) - *Machine Learning* - McGraw Hill, N.Y.

[Randell 1979] Randell B. (1979) - An Annotated Bibliography on the Origins of Digital Computers - *Annals of the History of Computing*, 1(2), pp.101-207.

[Clements 2000] Clements A. (2000) - *Principles of Computer Hardware* - Oxford Univ. Press.

[Wyatt 2002] Wyatt A. L. (2002) - *Using Assembly Language* - Que Publishing, Indianapolis.

[Wilcox 1990] Wilcox A.D. (1990) - *Engineering Design for Electrical Engineers* - Prentice-Hall.

[Emmerich 2000] Emmerich W. (2000) - *Engineering Distributed Objects* - John Wiley and Sons, N.Y.

[Wulf et al, 1974] Wulf W., Cohen E., Corwin W., Jones A., Levin R., Pierson C., Pollack F. (June 1974) - Hydra: The Kernel of a Multiprocessor Operating System - *Communications of the ACM*, pp. 337-345.

[Ding 2000] Ding-Zhu Du, Ker-I Ko (2000) - *Theory of Computational Complexity* - Wiley-Interscience, N.Y.

[Ottenstein 1984] Ottenstein K.J., Ottenstein L.M. (1984) - The Program Dependence Graph in a Software Development Environment - *ACM SIGPLAN Notices* 19(5).

[Rocchi 2002] Rocchi P. (August 2002) - On the Cruelty of Teaching Informatics - *Proc. of Informatics Curricula, Teaching Methods and Best Practice (ICTEM)*, pp. 1-11.

[Rocchi 2000] Rocchi P. (2000) - *Technology + Culture = Software* - IOS Press, Amsterdam.