Génie Logiciel et Intelligence Artificielle

H.-L. Bouziane[1], D. Delahaye[1], C. Dony[1], M. Huchard[1], C. Nebut[1], P. Reitz[1], A.-D. Seriai[1], C. Tibermacine[1], A. Etien[2], N. Anquetil[2], C. Urtado[3], S. Vauttier[3], J.-R. Falleri[4], L. du Bousquet[5], I. Alloui[6], and F. Vernier[6]

[1]Équipe MaREL, LIRMM, Université de Montpellier, CNRS, Montpellier
[2]Équipe RMoD, Cristal, Inria, Université de Lille, CNRS, Lille
[3]EuroMov Digital Health in Motion, Univ. Montpellier, IMT Mines Ales, Ales
[4]Équipe Progress, LaBRI, CNRS et ENSEIRB, Bordeaux
[5]LIG, Université Grenoble Alpes, Grenoble
[6]LISTIC, Université Savoie Mont Blanc, Annecy

1 Introduction

Les différentes branches de l'Intelligence Artificielle (IA) offrent de nombreuses opportunités de développement dans de très nombreux secteurs de l'activité humaine, dont certains ont été identifiés dans le rapport Villani [31] comme prioritaires tels que l'énergie, la santé, les transports et la sécurité. En particulier, la science du logiciel en tire de nombreux bénéfices depuis longtemps. Dans la communauté francophone plus spécifiquement, on peut se rappeler que la conférence LMO (Langages et Modèles à Objets) a regroupé, dès son origine, des recherches en représentation des connaissances et en programmation, et a été le creuset de travaux communs aux chercheurs de ces deux domaines [6]. L'IA peut plus largement apporter des solutions pour la réduction des coûts et des temps de développement et de maintenance, ainsi que pour l'amélioration de la qualité, dont la vérification.

Ces dernières années ont été marquées par un essor de l'IA, que ce soit dans les discours grand public ou dans les travaux de recherche. Au cœur de cet essor, on trouve différentes techniques dont les plus visibles actuellement sont les techniques d'apprentissage automatique (*machine learning*), avec en particulier l'apprentissage profond (*deep learning*), qui ont tendance à occulter la largeur et la richesse de ce domaine. Cette prégnance de l'IA dans toutes les activités nous oblige à nous positionner en tant que communauté du Génie Logiciel (GL) afin d'identifier les opportunités et les interactions entre GL et IA qui s'offrent à nous dans les années qui viennent. Ce positionnement sera également l'occasion de faire le point sur les branches de l'IA qui sont les plus pertinentes pour notre communauté et de rappeler brièvement les travaux qui ont pu être entrepris en GL en interaction avec le domaine de l'IA.

Les domaines du GL et de l'IA partagent plusieurs secteurs communs. Les systèmes multi-agents et les systèmes à composants partagent de nombreuses problématiques comme l'organisation en petites entités autonomes et communicantes, la résolution distribuée de problèmes, l'auto-adaptation ou encore la fiabilité et la tolérance aux fautes, avec de la fertilisation croisée entre les domaines [2]. La définition de modèles et d'architectures de logiciels compréhensibles s'appuie sur des techniques

de représentation des connaissances, telles que les graphes de connaissances ou les ontologies [7]. L'interopérabilité entre services web ou l'automatisation des transformations de modèles ou de méta-modèles s'appuient sur des techniques d'alignement de schémas ou d'ontologies [17]. La génération automatique de compositions de services utilise des techniques de planification [4]. La réparation automatique d'architectures logicielles s'appuie sur la résolution heuristique de problèmes sous contraintes [10]. Lors de processus d'évolution, la propagation automatique de changements permettant de restaurer la consistance des architectures logicielles est inférée par model-checking [18]. Les techniques de traitement de la langue naturelle sont couramment appliquées par exemple à l'ingénierie des exigences [5], à l'analyse des identificateurs [11], ou encore aux termes apparaissant dans la documentation ou les rapports de *bugs* [28]. La programmation par contraintes est utilisée dans la localisation de fautes [3] ou pour la génération de modèles conformes à un méta-modèle [12]. On peut noter aussi que de plus en plus de travaux utilisent de l'apprentissage statistique. Pour en donner quelques exemples, Tufano et al. montrent qu'il est possible, dans un contexte restreint, d'apprendre des modifications de code pour le *refactoring* ou la correction de *bugs* à partir des codes sources et des *pull requests* [29] ; Zhanh et al. [35] identifient des instructions suspectes dans du code ; Palmerino et al. [22] utilisent la régression linéaire multiple pour les systèmes auto-adaptatifs.

L'apprentissage statistique prend actuellement une grande place grâce aux nouvelles capacités des ordinateurs. Cependant, les deux courants de l'IA (symbolique et statistique) ont chacun leur rôle à jouer. Les approches symboliques et logiques sont utiles pour leur aspect déductif et leur prise en compte des aspects conceptuels des domaines étudiés (ici les données particulières du GL). Les approches statistiques sont utiles pour leur efficacité et leur aspect prédictif. Les limites des approches symboliques tiennent dans le besoin d'expliciter les connaissances et les règles, et de mener des raisonnements parfois coûteux. Les approches statistiques ont leurs limites également comme le fait de prendre peu en compte la connaissance d'un domaine et la structure des informations. On leur reproche aussi leur côté « boîte noire » et la difficulté, voire l'impossibilité, à expliquer les résultats. L'entraînement à partir d'exemples crée des biais dans l'apprentissage, qui sont potentiellement aussi graves pour les pratiques et constructions logicielles que pour les aspects sociaux [21]. Des travaux s'intéressent à certains des problèmes posés, comme le test des algorithmes à base d'apprentissage profond dans [16], les vulnérabilités [33], la prise en compte des structures comme les arbres de syntaxe abstraite et la tentative d'apporter une explication [14]. Frank van Hermelen, lors d'un exposé invité à la conférence EGC 2018 [1] développe par ailleurs une vision consistant à proposer des schémas de combinaison des deux courants (symbolique et statistique), plutôt que de les opposer. L'approche d'extraction d'architecture présentée dans [27] est un exemple d'une telle combinaison entre une approche de classification statistique (regroupement hiérarchique basé sur une métrique de similarité) et une approche symbolique (l'analyse formelle de concepts).

Du côté de la communauté IA, le développement de systèmes dans des secteurs tels que ceux de l'énergie ou la santé nécessite de plus en plus de compétences en ingénierie du logiciel. Ceci est d'autant plus vrai que les systèmes requièrent des architectures logicielles distribuées, dynamiques et évoluant dans le temps en fonction des besoins humains. D'autre part, les développeurs des systèmes ou applications IA ont besoin de méthodes, langages et outils pour le développement de ces systèmes.

Dans cet article de positionnement, nous présenterons tout d'abord ce que l'IA peut offrir au domaine du GL (section 2), puis nous verrons comment les techniques du GL peuvent servir en retour le domaine de l'IA (section 3), notamment dans le cadre de systèmes intégrant des composants produits par des techniques d'IA.

1. « Combining Learning and Reasoning : New Challenges for Knowledge Graphs », exposé disponible à l'adresse suivante : `http://www.canalc2.tv/video/15255`.

2 IA pour le GL

2.1 Sûreté de fonctionnement

L'IA symbolique a déjà beaucoup apporté au domaine de la sûreté de fonctionnement, lorsque l'on développe notamment des systèmes nécessitant une plus grande garantie de confiance, comme les systèmes critiques (systèmes dont les défaillances peuvent avoir des conséquences dramatiques en termes de pertes humaines, financières, ou encore sur l'environnement). Dans le domaine des méthodes formelles, par exemple, un certain nombre d'outils développés et utilisés reposent sur des méthodes qui visent à automatiser au maximum le processus de vérification. Parmi ces méthodes, on trouvera en particulier la déduction automatique (dans différentes logiques et différentes théories) et le *model-checking*. Ces méthodes sont souvent privilégiées par les outils utilisés en milieu industriel afin de minimiser les coûts de développement. C'est le cas, par exemple, de l'Atelier B (qui repose sur la méthode B [1]), qui intègre à la fois des techniques de preuve automatique et de *model-checking*.

Avec l'essor de l'apprentissage automatique, on peut raisonnablement se demander comment cet ensemble de nouvelles techniques devenues parfaitement effectives peuvent contribuer au domaine de la sûreté de fonctionnement en ajoutant notamment une composante prédiction à des méthodes où le calcul est prédominant. Un certain nombre de travaux ont déjà été entrepris dans ce domaine. Aujourd'hui, il semble que l'apprentissage automatique ne produit pas forcément de très bons résultats lorsqu'il est appliqué dans des contextes très formels, ce qui dès lors en fait un défi tout aussi intéressant que de taille pour l'avenir aussi bien pour la communauté GL que pour la communauté IA. Par exemple, dans le domaine de la preuve interactive (permettant de démontrer formellement la correction de programmes, par exemple), les expérimentations récentes dans des systèmes comme Isabelle [19] ou Coq [8] montrent des difficultés de l'apprentissage automatique à guider l'utilisateur dans la preuve, c'est-à-dire lorsqu'il faut lui recommander une tactique de preuve. Les difficultés sont encore plus grandes lorsque la tactique de preuve est paramétrée, car actuellement il est impossible de reconstruire ces paramètres par apprentissage automatique. On est donc loin d'une situation satisfaisante et il reste encore du chemin avant d'avoir un outil de recommandation de tactiques effectif.

De même, dans le domaine de la preuve automatique, l'apprentissage automatique ne permet pas de déduire la preuve automatiquement et il ne faut donc pas aborder le problème de front. En revanche, si on est moins ambitieux, on peut utiliser l'apprentissage automatique pour aider la recherche de preuve. Par exemple, l'apprentissage automatique peut permettre de sélectionner les axiomes d'une théorie nécessaires pour démontrer une certaine formule. Cette sélection, appelée sélection de prémisses, est importante car elle permet de réduire l'espace de recherche de preuve, et donc d'améliorer significativement la recherche de preuve dans certains cas. Les premières expérimentations remontent à il y a plus de 10 ans [30], avec la difficulté de définir des ensembles de caractéristiques pertinentes pour caractériser les énoncés mathématiques dans les théories considérées [15]. Ces caractéristiques sont principalement des symboles, mais peuvent également être des types, des sous-termes, etc. L'apprentissage profond est une piste plus récente, puisque l'une des premières expériences date d'il y a tout juste 4 ans [13], et qui permet de s'affranchir de cette difficulté car les caractéristiques seront automatiquement créées par le réseau de neurones lorsqu'il apprendra. Des expérimentations très récentes [9] montrent que cette technique donne des résultats prometteurs dans des théories réputées difficiles en preuve automatique, comme la géométrie par exemple.

Ainsi, aujourd'hui, il semble raisonnable d'utiliser l'apprentissage automatique comme aide aux méthodes formelles, mais certaines difficultés subsistent si l'on souhaite le faire apparaître comme un acteur principal du processus de développement. Ces difficultés sont liées à un certain nombre d'éléments que l'apprentissage automatique ne prend pas en compte, ou mal. Par exemple, si les

données d'entrée sont des formules de logique du premier ordre, on aimerait bien que la structure arborescente, ainsi que les liaisons des variables, soient analysées et exploitées, plutôt que de voir ces formules comme de simples chaînes de caractères ; le *GraphDL* [34] est une piste prometteuse.

2.2 Maintenance et évolution

Depuis quelques années, on voit apparaître de grandes bases de code ouvert, comme GitHub.com, GitLab.com, Maven Central ou Software Heritage. Ces bases de code ne cessent de grandir en taille. Elles fournissent de précieuses informations sur l'évolution des systèmes à code source ouvert, leurs différentes versions, voire même, dans certaines bases, les descriptions des modifications apportées à ces systèmes entre les versions (les messages des *commits*, les journaux de modifications ou *changelogs*, etc.). Toutes ces informations peuvent être utilisées dans l'apprentissage automatique pour entraîner des classificateurs permettant de résoudre des problèmes de GL comme, par exemple : (1) reproduire l'évolution de programmes utilisant une certaine bibliothèque vers une autre bibliothèque, en synthétisant des adaptateurs [20] ou en générant des opérations de *refactoring* ; (2) migrer un projet d'une version d'une bibliothèque à une autre, voire même d'un langage ou d'un paradigme à un autre, par exemple du paradigme objet vers le paradigme Workflow [26] ; (3) restructurer une grande application à objets vers un système de modules.

Un des défis à relever ici est celui de l'exploitation de ces grandes collections de données de projets à code source ouvert, qui incluent beaucoup de connaissances enfouies, pour extraire et apprendre les bonnes pratiques suivies par les développeurs afin d'assister l'activité de ré-ingénierie du code. Toute la difficulté dans ce défi réside dans la définition du modèle adéquat et de l'optimisation de ce dernier, avec les paramètres précis, pour réaliser l'apprentissage automatique mentionné ci-dessus. L'autre point difficile à traiter concerne le pré-traitement à réaliser sur ces collections de données pour, entre autres, éliminer tout le bruit qui peut exister dans ces données, comme les mauvaises pratiques de développement et d'évolution. Il est intéressant de noter que ce défi est directement connecté à d'anciens défis du GL. Par exemple, au début des années 1970, Chuck Rich, Dick Waters et Howard Shrobe, inspirés par un certain nombre d'autres personnes du laboratoire d'IA du MIT (par exemple, Terry Winograd, Carl Hewitt et Gerry Sussman), ont proposé l'idée d'un *apprenti programmeur* [23], un assistant intelligent qui aiderait un programmeur à écrire, déboguer et faire évoluer des logiciels. La présence d'un large sous-ensemble de tous les logiciels du monde disponible en ligne dans des dépôts combinée aux avancées récentes de l'apprentissage automatique, permettant de faciliter l'acquisition de connaissances par fouille de données, devrait permettre de faire un grand pas pour faire de cette vision une réalité. Un tel assistant devra également passer par le développement d'interfaces intelligentes, qui devront apprendre des habitudes de développement de l'utilisateur et non plus du code lui-même, dans la lignée de travaux comme [24], où l'on va chercher à automatiser certaines tâches répétitives.

Certaines méthodes d'IA permettent de faire de la prédiction automatique. Cependant, en évolution logicielle, si la modification à apporter est trop complexe, le développeur refusera son automatisation. En effet, certains *refactorings* ne sont pas utilisés par les développeurs par manque de confiance dans les outils. D'autre part, un certain nombre d'approches peuvent faire des recommandations et donc proposer plusieurs possibilités au développeur qui devra alors choisir. Les approches sont alors semi-automatiques ou correspondent simplement à un support outillé sans être totalement automatisé. Le défi à relever ici est donc celui de mobiliser des techniques d'IA capables de prendre en compte l'intervention humaine. Enfin, de façon plus générale, un défi consisterait à recenser les problèmes typiques de maintenance et d'évolution et à identifier, parmi les techniques d'IA, celles qui sont les plus adaptées pour leur apporter des solutions.

3 GL pour l'IA

3.1 Développement de systèmes intégrant de l'IA

Certaines approches d'apprentissage machine produisent des « composants appris » et non plus spécifiés, ce qui impose de revisiter l'ensemble des étapes du cycle de développement : récolte des exigences, architecture/conception, codage, validation, exécution et maintenance/évolution. L'intégration de composants de ce type a des impacts sur toutes les propriétés fonctionnelles et non fonctionnelles (sûreté de fonctionnement, sécurité, performance, utilisabilité, maintenabilité, etc.). D'une certaine façon, la qualité du résultat final (tous points de vue confondus) va beaucoup dépendre du travail en amont (récolte des exigences et conception), de ce que le composant « intelligent » pourra « justifier » en plus des sorties (explications, transparences, etc.), de la manière dont le système est construit, etc. La construction d'un logiciel/système qui comprend un composant appris impose un changement de point de vue. En effet, ces nouveaux composants ont des limitations :

— ils ne sont pas forcément capables de donner un résultat,
— ils ne sont pas forcément déterministes,

et peuvent évoluer et changer de comportement :

— ils peuvent continuer à apprendre,
— ils peuvent s'adapter à un contexte changeant.

De ce fait, la façon de construire le logiciel (spécification, développement, test, etc.) doit être repensée. Par exemple, on peut imaginer proposer des « patrons/styles architecturaux » ou des points de questionnement :

— Comment prendre en compte l'utilisateur dans la boucle ?
— Comment exploiter les explications des algorithmes d'IA ?
— Comment s'assurer à la volée de la pertinence des entrées du composant ?
— Comment gérer un modèle qui apprend à la volée ?

Des stratégies de *monitoring* doivent être mises en œuvre pour suivre le comportement du système logiciel pendant son exécution.

Revisiter l'ingénierie dirigée par les modèles et la production de *Domain Specific Languages* (DSLs) semblent être des pistes intéressantes pour le développement de ces systèmes. Un des défis réside en la composition et la coordination de différents DSLs. Des approches sémantiques issues de l'IA telles que les ontologies sont justement complémentaires des DSLs, puisque dans les deux cas, il s'agit d'expliciter les notions d'un domaine.

3.2 Validation et vérification

Ces dernières années, l'apprentissage automatique a concentré l'attention du monde de la recherche académique, mais aussi celle de l'industrie. Tous les secteurs sont concernés aussi bien critiques comme l'automobile, l'industrie, la finance ou la santé, que les secteurs a priori non critiques. Si l'on dispose aujourd'hui d'une large palette de méthodes et d'outils pour valider ou vérifier les programmes que l'être humain écrit en se basant notamment sur des spécifications, on manque d'approches pour établir la correction de programmes/composants appris. Un certain nombre de métriques sont utilisées afin de déterminer la qualité d'un composant appris, cependant ces dernières restent souvent d'ordre fonctionnel, comme le taux de bonnes ou mauvaises réponses sur un panel d'informations. D'autres méthodes permettent d'attester la qualité des données d'apprentissage [25], mais cela ne permet pas d'extrapoler la qualité du composant final.

La situation est particulièrement problématique dans le cas des systèmes critiques. Dans l'automobile par exemple, l'apprentissage automatique combiné à la puissance de calcul informatique qui ne cesse de croître permet d'identifier en temps réel les éventuels obstacles qui se présentent sur une route, une compétence indispensable pour développer des véhicules autonomes. Mais ce type de détection n'est pas infaillible et des accidents dramatiques sont déjà à déplorer (voir par exemple l'accident mortel en 2018 impliquant un véhicule autonome Uber et un piéton [32]).

Aujourd'hui, on ne sait pas spécifier formellement les résultats produits par les algorithmes d'apprentissage automatique, dont on ne peut pas toujours expliquer le comportement et dont les résultats ne sont pas toujours reproductibles (deux apprentissages sur le même jeu de données peuvent donner deux systèmes différents). Faute de pouvoir proposer approche certifiée, il faudra probablement opter pour une approche certifiante pour les systèmes critiques. Elle consiste à valider/vérifier le résultat au coup par coup, par exemple avec des approches de *monitoring*. D'une façon générale, du point de vue de la validation/vérification, il est indispensable que la méthode d'apprentissage automatique puisse fournir un certificat, qui permettra de vérifier le résultat et qui est nécessaire lorsqu'il est impossible de vérifier le résultat seul. Ce certificat peut être vu comme une explication et c'est justement ce que les algorithmes d'apprentissage sont incapables de bien produire aujourd'hui.

À noter que même pour des systèmes a priori non critiques, l'intégration de l'IA peut s'avérer problématique en termes de validation. Ainsi, l'utilisation de modèles d'apprentissage à la volée dans un logiciel permet par exemple d'adapter les comportements du systèmes aux comportements et attentes des utilisateurs et de son environnement. En d'autres termes, le logiciel doit être capable d'affiner ses services en fonction de besoins non exprimés (donc non spécifiés), mais déduits de son utilisation. Dans cette situation :

- Comment valider un système dont l'une des caractéristiques est d'évoluer naturellement ?
- Comment garantir, au cours de sa vie et son évolution, qu'il rende le service attendu par l'utilisateur, avec une qualité au moins identique ?
- Comment maintenir et faire évoluer un système depuis sa conception initiale ?
- Comment intégrer et garantir que ce qu'il a appris ne soit pas dégradé par une maintenance ou une évolution du système ?

4 Conclusion

Dans cette courte et partielle synthèse et selon notre point de vue, nous avons cherché à montrer comment l'IA peut soutenir nos travaux en GL dans ses efforts pour proposer des méthodes, des pratiques et des outils pour maîtriser le développement et la maintenance (voir la section 2). Nous avons également chercher à montrer que le GL pouvait en retour apporter ses méthodes à l'IA (voir la section 3), notamment sur la confiance, et que le GL devait se réinventer dans le développement de systèmes intégrant des composants produits par apprentissage automatique. Il nous a semblé aussi important de ne pas restreindre l'IA aux techniques d'apprentissage (même si ces dernières sont clairement à l'origine de l'engouement actuel pour le domaine de l'IA), et aussi de ne pas opposer IA symbolique et statistique en montrant que ces deux courants pouvaient se compléter de manière générale et dans le cadre de travaux de la communauté GL en particulier. Pour toutes ces raisons, il nous semble opportun de proposer la création d'un groupe de travail du GDR GPL, qui nous permettrait de réfléchir aux différents défis et questions de recherche s'offrant à nous à l'intersection du GL et de l'IA. Du fait de l'interaction forte avec le domaine de l'IA, il nous semble également important de favoriser les collaborations avec certains groupes de travail du GDR IA (des discussions sont déjà en cours sur le sujet et seront concrétisées ultérieurement si ce groupe de travail est créé).

Références

[1] J.-R. Abrial. *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195.

[2] J.-P. Arcangeli, V. Noel, and F. Migeon. Software Architectures and Multi-Agent Systems. In *Software Architectures*, volume 2, chapter 5, pages 171–208. Wiley, mai 2014.

[3] N. Aribi, M. Maamar, N. Lazaar, Y. Lebbah, and S. Loudni. Multiple Fault Localization Using Constraint Programming and Pattern Mining. In *ICTAI 2017*, pages 860–867, 2017.

[4] A. Bekkouche, S. M. Benslimane, M. Huchard, C. Tibermacine, H. Fethallah, and M. Mohammed. QoS-Aware Optimal and Automated Semantic Web Service Composition With User's Constraints. *Service Oriented Computing and Applications*, 11(2) :183–201, 2017.

[5] E. E. Bella, M. Gervais, R. Bendraou, L. Wouters, and A. Koudri. Semi-Supervised Approach for Recovering Traceability Links in Complex Systems. In *ICECCS 2018,*, pages 193–196, 2018.

[6] B. Carré, R. Ducournau, J. Euzenat, A. Napoli, and F. Rechenmann. Classification et objets : programmation ou représentation ? In *5e journ. nat. PRC-GDR IA*, pages 213–237, Feb. 1995.

[7] C. Coral, R. Francisco, and P. Mario. *Ontologies for Software Engineering and Software Technology*. Springer-Verlag, Berlin, Heidelberg, 2006.

[8] D. Delahaye and A. Izard. Utilisation de techniques d'apprentissage automatique pour l'aide à la preuve dans le système Coq. Technical report, Université de Montpellier, 2019.

[9] D. Delahaye and B. Lemoine. Déduction automatique en géométrie en utilisant l'apprentissage profond. Technical report, Université de Montpellier, 2019.

[10] N. Desnos, M. Huchard, G. Tremblay, C. Urtado, and S. Vauttier. Search-Based Many-To-One Component Substitution. *Journal of Software Maintenance and Evolution : Research and Practice.*, 20(5) :321–344, September/October 2008.

[11] J. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao. Automatic Extraction of a WordNet-Like Identifier Network from Software. In *ICPC'10*, pages 4–13, 2010.

[12] A. Ferdjoukh, A. Baert, E. Bourreau, A. Chateau, R. Coletta, and C. Nebut. Instantiation of Meta-models Constrained with OCL - A CSP Approach. In *MODELSWARD 2015*, pages 213–222, 2015.

[13] G. Irving, C. Szegedy, A. A. Alemi, N. Eén, F. Chollet, and J. Urban. DeepMath – Deep Sequence Models for Premise Selection. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 2235–2243, Barcelona (Spain), Dec. 2016.

[14] L. Jiang, H. Liu, and H. Jiang. Machine Learning Based Automated Method Name Recommendation : How Far Are We. In *ASE 2019*, 2019.

[15] C. Kaliszyk, J. Urban, and J. Vyskočil. Efficient Semantic Features for Automated Reasoning over Large Theories. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3084–3090, Buenos Aires (Argentina), July 2015. AAAI Press.

[16] J. Kim, R. Feldt, and S. Yoo. Guiding Deep Learning System Testing using Surprise Adequacy. In *ICSE 2019*, pages 1039–1049, 2019.

[17] L. Lafi, J. Feki, and S. Hammoudi. Metamodel Matching Techniques : Review, Comparison and Evaluation. *IJISMD*, 5(2) :70–94, 2014.

[18] A. Mokni, M. Huchard, C. Urtado, S. Vauttier, and H. Y. Zhang. A Formal Approach for Managing Component-Based Architecture Evolution. *Science of Computer Programming*, (127) :24–49, 2016.

[19] Y. Nagashima and Y. He. PaMpeR : Proof Method Recommendation System for Isabelle/HOL. In *ASE 2018*, pages 362–372, 2018.

[20] M. Nita and D. Notkin. Using Twinning to Adapt Programs to Alternative APIs. In *ICSE'10 - Volume 1*, ICSE '10, pages 205–214. ACM, 2010.

[21] C. O'Neil. *Algorithmes : la bombe à retardement* . Les Arènes, Paris, 2018.

[22] J. Palmerino, Q. Yu, T. Desell, and D. Krutz. Improving the Decision-Making Process of Self-Adaptive Systems by Accounting for Tactic Volatility. In *ICSE 2019*, 2019.

[23] C. Rich and R. C. Waters. The Programmer's Apprentice : A Research Overview. *Computer*, 21(11) :10–25, Nov. 1988.

[24] J. Ruvini and C. Dony. Learning Users' Habits to Automate Repetitive Tasks. In *Your Wish is My Command*, The Morgan Kaufmann series in interactive technologies, pages 271–296. Morgan Kaufmann / Elsevier, 2001.

[25] S. Sadiq, N. K. Yeganeh, and M. Indulska. 20 Years of Data Quality Research : Themes, Trends and Synergies. In *Australasian Database Conference (ADC)*, volume 115, pages 153–162, Darlinghurst, Australia, 2011. Australian Computer Society, Inc.

[26] A. Selmadji, A. Seriai, H. Bouziane, and C. Dony. From Object-Oriented to Workflow : Refactoring of OO Applications into Workflows for an Efficient Resources Management in the Cloud. In *ENASE 2018, Revised Selected Papers*, pages 186–214, 2018.

[27] A. Shatnawi, A.-D. Seriai, and H. Sahraoui. Recovering Software Product Line Architecture of a Family of Object-Oriented Product Variants. *Journal of Systems and Software*, 131 :325–346, Sept. 2017.

[28] Y. Tian and D. Lo. A Comparative Study on the Effectiveness of Part-of-Speech Tagging Techniques on Bug Reports. In *SANER 2015*, pages 570–574, 2015.

[29] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk. On Learning Meaningful Code Changes via Neural Machine Translation. In *ICSE 2019*, pages 25–36, 2019.

[30] J. Urban, G. Sutcliffe, P. Pudlák, and J. Vyskočil. MaLARea SG1 – Machine Learner for Automated Reasoning with Semantic Guidance. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *LNCS*, pages 441–456, Sydney (Australia), Aug. 2008. Springer.

[31] C. Villani, M. Schoenauer, Y. Bonnet, C. Berthet, A.-C. Cornut, F. Levin, and B. Rondepierre. *Donner un sens à l'intelligence artificielle : Pour une stratégie nationale et européenne*. 03 2018.

[32] D. Wakabayashi. Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. *New York Times*, 19th March 2018. https ://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html [Retrieved April, 2020].

[33] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang. Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing. In *ICSE 2019*, pages 1245–1256, 2019.

[34] Z. Zhang, P. Cui, and W. Zhu. Deep Learning on Graphs : A Survey. *CoRR*, abs/1812.04202, 2018.

[35] Z. Zhang, Y. Lei, X. Mao, and P. Li. CNN-FL : An Effective Approach for Localizing Faults using Convolutional Neural Networks. In *SANER 2019*, pages 445–455, 2019.