Matrice d'auto-évaluation des compétences en programmation

		A1 Utilisateur élémentaire	A2 Utilisateur élémentaire	B1 Utilisateur indépendant	B2 Utilisateur indépendant	C1 Utilisateur expérimenté	C2 Utilisateur expérimenté
Écrire	Écrire du code	Je peux produire une implémentation correcte pour une fonction simple, à partir d'une spécification bien définie du comportement et de l'interface souhaités, sans l'aide d'autrui.	Je peux déterminer une interface appropriée et produire une implémentation correcte, à partir d'une spécification vague pour une fonction simple, sans l'aide d'autrui. Je peux décomposer une spécification de fonction complexe en fonctions plus petites.	Je peux estimer les coûts en espace et en temps de mon code lors de son exécution. Je peux comparer empiriquement différentes implémentations de la même spécification de fonction à l'aide de métriques bien définies, notamment le temps d'exécution et l'empreinte mémoire. J'exprime des invariants dans mon code en utilisant des préconditions, des assertions et des post-conditions. J'utilise des bouchons ("stubs") pour gagner en flexibilité sur l'ordre d'implémentation.	J'utilise le typage et les interfaces de manière délibérée et productive pour structurer et planifier à l'avance mon activité de codage. Je peux concevoir et mettre en œuvre moi-même des programmes entiers à partir de spécifications bien définies sur les entrées et sorties externes. Je tente systématiquement de généraliser les fonctions pour augmenter leur réutilisabilité.	Je peux systématiquement reconnaître des exigences incohérentes ou contradictoires dans les spécifications. Je peux décomposer une architecture de programme complexe en composants plus petits qui peuvent être mis en œuvre séparément, y compris par d'autres personnes. Je peux utiliser des langages dédiés (embarqués ou non) ou des modèles de métaprogrammation existants pour augmenter ma productivité.	Je peux reconnaître de manière fiable quand une spécification est incomplète de manière intentionnelle ou non. Je peux exploiter une spécification incomplète pour augmenter ma productivité de manière non triviale. Je peux concevoir de nouveaux langages dédiés (embarqués ou non) ou créer de nouveaux modèles de métaprogrammation pour augmenter ma productivité et celle des autres.
	Refactoriser	Je peux adapter mon code lorsque je reçois de petites modifications dans sa spécification sans le réécrire entièrement, à condition de savoir que la modification est incrémentielle. Je peux modifier mon propre code grâce aux instructions détaillées d'une personne plus expérimentée.	Je peux déterminer moi-même si un petit changement de spécification est incrémentiel ou nécessite une refactorisation importante. Je peux modifier mon propre code grâce aux instructions vagues d'une personne plus expérimentée.	Je peux dériver une stratégie de refactorisation sur mon propre code, compte tenu de changements relativement mineurs dans les spécifications. Je peux modifier le code d'autrui grâce aux instructions précises d'une personne déjà familiarisée avec le code.	Je peux prédire avec précision l'effort nécessaire pour adapter ma propre base de code à une nouvelle spécification. Je peux suivre une stratégie de refactorisation existante sur le code de quelqu'un d'autre. Je peux assumer l'entière responsabilité de l'intégration du correctif de quelqu'un d'autre dans mon propre code.	Je peux effectuer de la rétro-ingénierie sur la base de code de quelqu'un d'autre avec l'aide de la spécification d'origine et prédire avec précision l'effort nécessaire pour l'adapter à une nouvelle spécification.	Je peux effectuer de la rétro-ingénierie sur la base de code de quelqu'un d'autre sans spécification originale et prédire avec précision l'effort nécessaire pour l'adapter à une nouvelle spécification.
	S'intégrer dans un système plus large	Je connais les points d'entrée et de terminaison dans le code que j'écris. Je peux utiliser les principaux canaux d'entrée/sortie de mon langage pour saisir et imprimer du texte simple et des nombres.	Je connais les mécanismes recommandés pour accepter depuis l'environnement d'exécution les options/paramètres d'un programme et signaler les erreurs, et je les utilise dans le code que j'écris.	Je peux déléguer des fonctions à un processus externe au moment de l'exécution. Je sais comment utiliser de manière productive la lecture en continu ("streaming") et la mise en mémoire tampon pour travailler sur de grands ensembles de données et les utiliser dans mon code. Je connais le principe de localité et je l'utilise pour adapter mes implémentations.	Je connais au moins une API pour la communication bidirectionnelle avec d'autres processus d'exécution. Je peux écrire du code client pour des protocoles Internet simples. Je connais les exigences les plus courantes en matière d'empaquetage et de redistribution d'au moins une plateforme et je les utilise dans mes propres projets. Je peux utiliser des modèles de programmation prédéterminés pour exploiter de manière productive le parallélisme des plateformes dans mon code.	Je peux implémenter des logiciels client et serveur pour des spécifications de protocole arbitraires. Je peux quantifier avec précision les coûts temporels et spatiaux de différents mécanismes de communication (par exemple, appels système, pipes, sockets). Je connais les architectures matérielles et je peux prédire le comportement des programmes séquentiels lors du changement du matériel sous-jacent. Je peux estimer la scalabilité de fragments de code parallèle sur une plateforme donnée.	Je connais la plupart des architectures logicielles utilisées avec les systèmes pour lesquels je développe. Je peux travailler avec des architectes système pour optimiser mutuellement ma propre architecture logicielle avec l'architecture système globale. Je connais la plupart des compromis coûts/avantages de conception et de productivité dans les systèmes pour lesquels je développe.
Comprendre	Réutiliser du code	Je peux assembler des fragments de programme en renommant des variables jusqu'à ce que l'ensemble devienne cohérent et compatible avec mon objectif.	À partir dune bibliothèque de fonctions pour la plupart pures et une documentation d'API détaillée, je peux réutiliser cette bibliothèque de manière productive dans mon code.	Je peux reconnaître quand le code existant nécessite une architecture globale particulière pour être réutilisé (par exemple une boucle d'événements). Je peux adapter mon propre code à l'avance aux exigences de plusieurs bibliothèques distinctes que je prévois de réutiliser.	Je peux reconnaître et extraire des composants réutilisables d'une base de code plus grande pour mon propre usage, même lorsque l'auteur original n'envisageait pas la réutilisabilité. Je peux empaqueter, documenter et distribuer une bibliothèque de logiciels pour que d'autres puissent la réutiliser. Je peux interfacer du code sans état provenant de différents langages de programmation.	Je peux systématiquement supprimer les contraintes du code existant qui ne sont pas imposées par la spécification, afin de maximiser sa généralité. Je peux lire et comprendre le code qui utilise les API les plus courantes dans mon domaine sans l'aide de leur documentation. Je peux interfacer du code provenant de différents langages de programmation avec une sémantique opérationnelle distincte.	Je peux découvrir et exploiter de manière fiable le comportement non documenté/involontaire de tout code écrit dans un langage que je comprends, y compris le code que je n'ai pas écrit moi-même.
	Expliquer / Discuter du code	Je peux lire le code que j'ai écrit et expliquer ce que je souhaite exprimer à une personne plus expérimentée que moi.	Je peux lire le code d'une personne d'un niveau similaire ou inférieur au mien et expliquer ce qu'il signifie. Je peux reconnaître et expliquer des incohérences simples entre la spécification et l'implémentation dans mon code ou dans le code d'une personne du même niveau que moi ou inférieur.	Je peux montrer et expliquer des fragments de code que j'écris dans un style impératif ou déclaratif à quelqu'un d'autre qui connaît un langage de programmation différent où le même style est prévalent, afin que cette personne puisse reproduire la même fonctionnalité dans le langage de son choix.	Je peux expliquer mes structures de données, mes algorithmes et mes modèles d'architecture à une autre personne en utilisant les termes standards de mon domaine, sans référence à mon code.	Je peux évaluer le niveau d'expertise de mon public et modifier ma façon de lui parler en conséquence. Je peux reconnaître quand une explication est trop ou insuffisamment détaillée pour un public donné et faire des retours en conséquence.	Je peux participer sans effort à toute conversation ou discussion sur le ou les langages que j'utilise et je suis à l'aise avec les constructions idiomatiques. Je peux proposer spontanément des exemples de code corrects et démonstratifs pour tous les concepts que j'ai besoin de partager avec les autres.
Interagir	Explorer, apprendre en autonomie	Je peux faire la distinction entre une invite de commande sur un shell et une invite de saisie pour un programme exécuté à partir de ce shell. Je peux suivre des tutoriels en ligne sans aide et atteindre le résultat prescrit. Je peux rechercher le texte des messages d'erreur courants et adapter l'expérience des autres à mes besoins.	Je peux faire la distinction entre les fonctionnalités générales d'un langage et les fonctionnalités spécifiques à une implémentation de langage particulière. Je peux lire le texte des messages d'erreur et comprendre leur signification sans aide extérieure.	Je peux lire la documentation de référence du ou des langages et APIs que j'utilise et m'y référer pour clarifier ma compréhension de fragments de code arbitraires. Je peux comprendre les concepts généraux dans des articles ou des présentations d'experts. Je peux tracer et déterminer qui est responsable d'un fragment de code arbitraire dans un système que j'utilise ou pour lequel je développe.	Je peux déduire le modèle opérationnel abstrait d'une API ou d'une bibliothèque à partir de son interface, sans accès à la documentation, et écrire de petits programmes de test pour tester si mon modèle est cohérent. Je peux reconnaître lorsqu'une documentation de référence pour un langage ou une API est incomplète ou contradictoire avec une implémentation de référence.	Je suis capable de lire et de comprendre la plupart des ouvrages spécialisés applicables aux langages que j'utilise. Je suis capable de reconnaître quand une innovation issue de la recherche universitaire est applicable à mon domaine et de l'adapter pour l'utiliser dans mes projets.	Je peux reconnaître et exposer des hypothèses tacites dans la littérature spécialisée dans mon domaine. Je peux reconnaître de manière fiable quand une présentation ou une description du résultat d'un programme est fausse ou trompeuse, sans tester explicitement.
	Maîtrise de l'environnement	Je peux utiliser un environnement de programmation commun et suivre étape par étape les processus courants pour tester/exécuter un programme.	Je peux intégrer mes fichiers sources dans un environnement de programmation qui automatise une grande partie de mon flux de travail de programmation. J'utilise la gestion de versions pour suivre mes progrès et annuler les modifications infructueuses.	J'exprime et utilise le suivi des dépendances dans mon environnement de programmation pour éviter les (re)traitements inutiles dans mes cycles de développement. Je peux utiliser différentes branches de développement dans la gestion de versions pour différentes tâches de programmation.	J'utilise différents processus pour différentes tâches de programmation, avec différents compromis entre les coûts de configuration initiale et les coûts de maintenance à long terme. Je peux entrer dans l'environnement d'une autre personne du même niveau que moi ou inférieur et apporter des contributions au code avec une formation minimale.	Je modifie mon environnement de programmation pour l'adapter à mon style personnel et je peux quantifier l'impact de ces changements sur ma productivité. Je peux utiliser de manière productive les environnements de programmation préférés d'au moins 80 % de tous les programmeurs de mon niveau ou inférieur.	Je peux reconnaître et quantifier de manière fiable les frictions entre les autres programmeurs et leur environnement de programmation. Je peux améliorer de manière mesurable la productivité de mes pairs en les aidant à adapter leur environnement à leur style personnel.
	Résoudre des problèmes	Je peux faire la différence entre les sorties correctes et incorrectes dans mes propres programmes. Je connais l'étiquette pour demander de l'aide à des personnes expertes dans mon domaine.	Je peux faire la différence de manière fiable entre une sortie incorrecte due à une entrée incorrecte et une sortie incorrecte due à une erreur de programme. Je peux réduire la localisation d'une erreur dans un programme complexe à un seul module ou fonction. Je peux isoler et corriger des bugs persistants ("Bohr bugs") dans mon propre code.	Je peux traduire les connaissances humaines ou les spécifications sur les invariants en assertions ou en contraintes de type dans mon propre code. Je peux inspecter l'état d'exécution d'un programme pour vérifier qu'il correspond à un invariant connu. J'écris et j'utilise des tests unitaires le cas échéant.	Je peux réduire une erreur d'un programme au programme le plus simple qui reproduise la même erreur. J'ai une ou plusieurs stratégies de travail pour tracer et corriger des bugs sensibles à l'effet de sonde ("Heisenbugs") dans un code que je peux comprendre. J'écris et j'utilise des tests de régression pour le code avec lequel je travaille directement.	Je peux concevoir des stratégies systématiques pour isoler et corriger des bugs en en apparence chaotiques ("Mandelbugs") dans un code que je peux comprendre. Je peux reconnaître un bug matériel dans un système piloté principalement par un logiciel que j'ai conçu.	Je peux tracer et déterminer avec précision la responsabilité de la plupart des comportements inattendus/indésirables dans les systèmes pour lesquels je développe. Je peux tracer et isoler les bugs matériels dans les systèmes où j'ai accès à toutes les sources logicielles.

Les notes explicatives sont disponibles en ligne à l'adresse suivante : http://science.raphael.poss.name/programming-levels.html