MASTER EN SCIENCES INFORMATIQUES

Les technologies persuasives au service de la qualité logicielle
revue critique de la littérature

Pierlot, Johan

Award date:
2020

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2019–2020

Les technologies persuasives au service de la qualité logicielle: revue critique de la littérature

Johan Pierlot

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Benoit Vanderose

Co-promoteur : Antoine Clarinval

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens avant tout à remercier Benoît Vanderose, mon promoteur, et Antoine Clarinval, mon co-promoteur, pour leurs précieux conseils, leur disponibilité et leur bonne humeur qui m'ont accompagné tout au long de la rédaction de mon mémoire.

Un grand merci à Laurence et Etienne pour leur participation à la relecture de ce travail.

Je tiens également à remercier le Monstre en spaghetti volant pour sa miséricorde et son infinie sagesse.

Enfin, un merci particulier à Mélissa pour son soutien indéfectible tout au long de mon cursus universitaire.

Et je ne peux terminer ces remerciements sans préciser que tout ceci a naturellement été fait pour Zack.

Résumé

Le développement logiciel est un processus délicat dont la qualité d'exécution doit être effective. D'une part, de nombreux outils existent pour aider les développeurs dans leur processus mais ces outils ne sont pas toujours utilisés à bon escient et ne produisent donc pas systématiquement les effets escomptés. Les technologies persuasives (TP), d'autre part, sont utilisées pour permettre d'obtenir un comportement cible. Alors que les outils d'aide au développement logiciel sont de plus en plus nombreux et que leur utilité est prometteuse concernant l'amélioration de la qualité logicielle (QL), l'engouement pour les TP semble plus s'orienter vers des secteurs comme la santé, l'écologie ou encore le marketing. Nous avons donc analysé le domaine de recherche couvrant l'application des TP dans le développement logiciel afin d'optimiser la QL. Nous avons réalisé un état de l'art en procédant à une revue de la littérature sur base d'un corpus de 35 publications.

Nous avons constaté que la gamification, qui reprend des principes de conception utilisés dans les TP, était rapidement devenue omniprésente dans la littérature. Son expérimentation et les résultats de ses effets sont plus répandus que ceux concernant les TP au sens plus large. Nous avons également pu dégager les tendances et les différences des TP utilisées dans les outils d'analyse de code (AC), principalement axées sur des principes de récompense et de motivation et les TP les plus efficaces des systèmes de recommandation (SR) qui sont plus axées sur des principes de similarité, de confiance et d'expertise. Ce domaine de recherche est encore jeune et, malgré des résultats positifs et globalement encourageants, il n'y a pas, à ce jour, d'outil (*i.e.* AC ou SR) dont l'efficacité sur le changement de comportement à long terme, concernant l'amélioration de la qualité du code, est validée.

Nous sommes arrivés à la conclusion que la maturité des outils existants doit encore évoluer. De plus, une cartographie de ces outils pourrait s'avérer efficace afin de mettre au point un framework d'utilisation basé sur l'assistant de comportement de Fogg [16]. Ceci dans le but de déterminer quel outil serait le plus adéquat dans une situation donnée.

Mots-Clés

Persuasion, qualité logicielle, développement logiciel

1. Introduction

La qualité du développement logiciel dépend de nombreux facteurs dont les développeurs devraient être informés. Pour les aider à produire un code de qualité, les environnements de développement intégré (IDE) modernes sont généralement dotés de nombreux outils. Ces outils, s'ils sont correctement maîtrisés, peuvent augmenter considérablement la qualité logicielle (QL). Cette amélioration qualitative du code peut notamment se faire grâce à l'utilisation d'un outil d'analyse de code (AC) qui va permettre une quantification des attributs internes d'un produit logiciel. Cette quantification se fera généralement par l'entremise de métriques logicielles (ML) grâce auxquelles un développeur pourra prendre conscience d'un problème potentiel et agir en conséquence. L'utilisation des technologies persuasives (TP) pour motiver les développeurs à réduire la toxicité de certaines métriques retournées participe donc à la réalisation d'un code de qualité.

Les développeurs peuvent également compter sur la présence dans les IDE de systèmes de recommandation (SR) qui offrent une information de qualité, au moment adéquat. Les SR peuvent donc être bénéfiques pour qui est désireux d'améliorer son code. Ils peuvent être de plusieurs natures. Certains, plus simples, se contentent d'afficher les commandes existantes d'un IDE pour faciliter le travail d'encodage du développeur là où d'autres vont agir de manière ponctuelle pour remonter une information importante ou proposer un conseil pertinent que l'utilisateur sera libre de traiter selon son envie ou ses besoins.

Malgré la multitude et la diversité de ces outils, il apparaît que les développeurs n'ont pas toujours conscience de leur existence. En outre, leur utilisation n'est pas toujours évidente. Le traitement des informations récoltées peut également s'avérer pénible, pour ne pas dire rébarbatif dans certains cas. Le recours à la persuasion pourrait donc mettre en évidence ces outils et motiver les développeurs à les utiliser davantage, toujours dans cette optique qualitative.

Nous avons réalisé un état de l'art dans le domaine des TP appliquées dans le développement logiciel pour en augmenter la qualité. Nous avons procédé à une revue de la littérature qui nous a permis de rassembler 35 publications sur base desquelles nous avons appliqué une méthodologie comparative afin de dégager les pistes de recherche possibles ainsi que les tendances actuelles en la matière. Cette analyse a notamment permis de mettre en avant les différences entre les AC et les SR en termes de TP employées. Elle montre également la nécessité d'approfondir les recherches dans le domaine ainsi que la possibilité de mettre en place un framework basé sur une cartographie des outils existants pour en faciliter la sélection opportune lors d'un développement logiciel.

Dans un premier temps, nous présentons le contexte général dans lequel ce travail s'inscrit en posant les bases des TP afin d'en comprendre les principes et découvrir ceux qui seront les plus adaptés dans ce domaine de recherche particulier. Nous abordons ensuite la QL afin de déterminer par l'entremise de quels indicateurs nous pouvons agir sur le comportement d'un développeur, ce qui nous amènera à la présentation des ML. Ces métriques sont couramment utilisées dans les AC pour permettre de quantifier les attributs de qualité d'un logiciel.

Nous développons ensuite les motivations qui nous ont poussés dans cette direction et nous détaillons les résultats obtenus suite à l'analyse comparative de notre corpus.

Enfin, nous apportons dans la dernière partie notre point de vue sur ces résultats. Nous essayons de faire ressortir les tendances, les risques et les défis à venir dans ce domaine de recherche et nous tentons de dégager des pistes de recherche possibles.

2. Contexte général

Dans le but de référencer les éléments clés des TP, nous reprenons dans un premier temps une vue d'ensemble des différentes études concernant les interfaces persuasives ainsi que leur domaine d'exploitation. Notre recherche se porte ensuite sur l'approche qualitative d'un produit logiciel et les outils et techniques disponibles pour optimiser la qualité du code produit. Ce faisant, nous pourrons étudier le lien sur l'application des interfaces persuasives dans le cadre plus spécifique du développement logiciel afin d'analyser l'existant dans ce domaine et d'explorer les possibilités offertes. Nous analysons également les critères persuasifs pertinents à mettre en oeuvre afin de favoriser de bonnes pratiques de programmation.

En effet, il existe plusieurs leviers sur lesquels agir afin d'optimiser la qualité du code d'un développeur. Cette recherche a pour but de les répertorier afin de dégager les meilleures pistes d'intégration de techniques de persuasion dans un IDE.

2.1 Les technologies persuasives

Les interactions homme-machine (IHM) sont devenues plus répandues que jamais, poussées par l'omniprésence technologique actuelle. Que ce soit dans la sphère privée ou professionnelle, il devient courant d'être confronté à un système informatique pour réaliser des tâches administratives, communiquer ou simplement se divertir. L'évolution des besoins des utilisateurs a donc naturellement incité à développer de nouvelles approches et de nouveaux outils de design et d'ergonomie, poussant toujours plus loin l'expérience utilisateur qui en découle. La persuasion dans les interfaces fait partie intégrante de ces pratiques et consiste à injecter des éléments persuasifs dans un système informatique afin d'agir sur le comportement des utilisateurs, de manière transparente ou coercitive.

Les interfaces persuasives et les concepts psychologiques sous-jacents qui les définissent constituent donc un carrefour entre deux mondes. D'un côté la technologie et son évolution exponentielle et de l'autre côté, la psychologie et l'étude des moyens cognitifs permettant d'arriver à une modification comportementale. C'est donc à l'intersection de ces deux courants que se situent les interfaces persuasives.

L'étude des TP est un sujet récent dans l'histoire de l'informatique et c'est véritablement à partir des années 2000 que celle-ci va prendre une ampleur grandissante. L'approche fonctionnelle d'un logiciel ou d'un site web ainsi que sa facilité d'utilisation ne suffisent plus, il convient maintenant d'y intégrer des indicateurs persuasifs

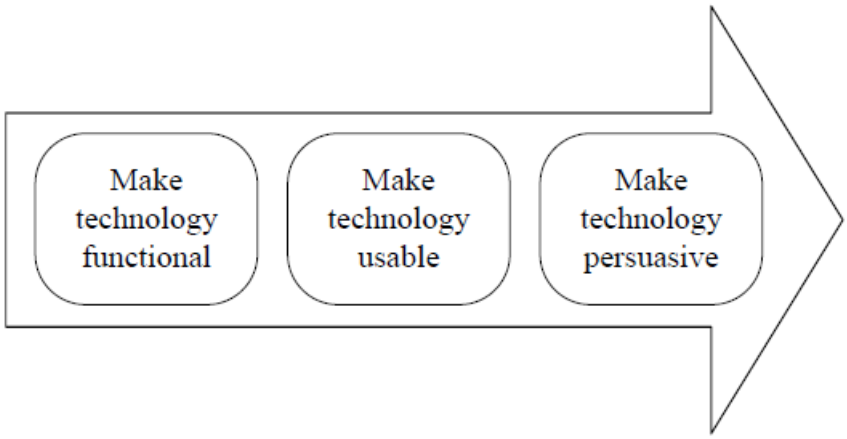

FIGURE 2.1 – Évolution du processus de création des interfaces [18]

dans le but de modifier le comportement ou l'attitude de l'utilisateur [18] (FIGURE 2.1).

2.1.1 Définition d'une technologie persuasive

Les recherches concernant l'impact des TP ont été développées dans les années nonante, principalement sous l'impulsion des travaux de Fogg [18] qui est à l'origine de la *captologie* (*i.e.* computers as persuasive technologies). Ce domaine de recherche définit les différents critères que doit posséder une technologie afin d'influencer le comportement de l'utilisateur.

Il faut bien insister sur la finalité de ces techniques qui est d'influencer les actes de l'utilisateur sur la durée par l'entremise de moyens cognitifs destinés à modifier durablement le comportement. Dans son travail, Fogg écarte les techniques coercitives ou les tromperies pour des raisons éthiques. Nous ferons de même dans ce document.

L'étude des moyens cognitifs nous amène donc à ne plus se focaliser uniquement sur le visuel et l'apparence d'une application ou d'un site, l'interface utilisateur (UI) mais bien d'amener l'utilisateur à modifier son attitude ou son comportement. Le concept d'expérience utilisateur (UX) évolue donc en fonction de l'évolution des technologies, de sa disponibilité et de sa présence de plus en plus grande mais également des besoins nouveaux et toujours plus diversifiés des utilisateurs [6].

D'après Fogg, il faut distinguer deux niveaux de persuasion :

- **Macro-persuasion** : les sites de vente en ligne représentent l'exemple parfait pour définir la macro-persuasion car leur seul objectif est de faire acheter l'utilisateur et de le pousser in fine à effectuer ses achats uniquement via le site en question. La persuasion constitue donc leur objectif premier.
- **Micro-persuasion** : la micro-persuasion dans les systèmes informatiques n'a pas cet objectif premier. Ces systèmes incorporent des éléments (*e.g* graphiques, sonores, etc.) qui vont agir comme des rappels et dont l'objectif est de changer la façon d'agir de l'utilisateur.

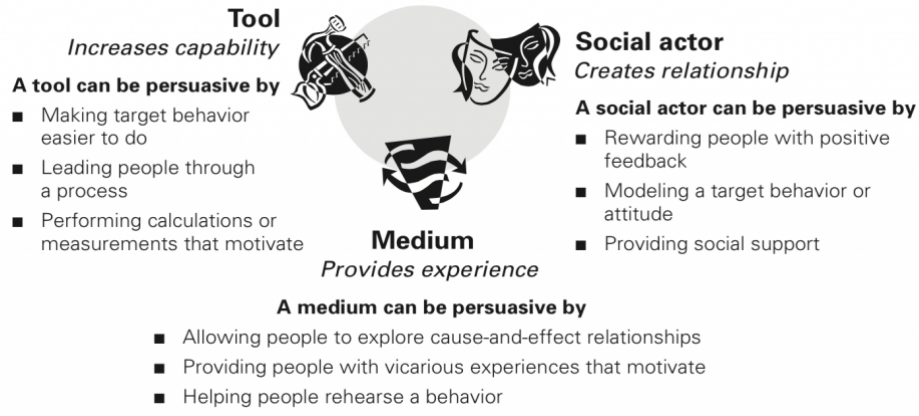

FIGURE 2.2 – Triade fonctionnelle de Fogg [18]

2.1.2 Mécanismes utilisés dans les technologies persuasives

Fogg définit un cadre conceptuel illustrant les trois rôles joués par la technologie informatique, appelé triade fonctionnelle (FIGURE 2.2). Ces rôles correspondent à la façon dont les gens réagissent à une technologie informatique. Le rôle d'*outil* pour augmenter la capacité des gens à atteindre un comportement cible. Le rôle de *média* pour créer des expériences persuasives et enfin le rôle *social* où la technologie est perçue comme un acteur social. Pour chacun de ces rôles, il reprend une liste des principes utilisés pour rendre persuasive une technologie (TABLE 2.1).

Prenons l'exemple du rôle social car nous le retrouverons notamment dans les critères de sélection des indicateurs persuasifs développés dans notre état de l'art. Le concept de cette approche consiste à présenter l'ordinateur comme une référence sociale en lui attribuant un rôle. En attribuant un rôle d'autorité à l'ordinateur (*e.g.* expert, professeur, conseiller, etc.), celui-ci sera perçu comme intelligent et influent (FIGURE 2.3 p.14). Fogg note qu'il faut cependant définir à qui s'adresse cette autorité car elle peut être perçue différemment selon l'utilisateur. Le rôle assigné à l'ordinateur peut également être celui d'un compétiteur, ce qui aura pour conséquence de changer le comportement de l'utilisateur. Il est donc important de connaître sa cible et même de proposer différents rôles au sein d'un même programme afin de donner à l'utilisateur la possibilité de sélectionner sa préférence.

En complément de ce cadre conceptuel et afin de renforcer l'efficacité des critères de persuasion, Fogg insiste sur l'importance de la crédibilité technologique et propose des principes de conception pour augmenter le pouvoir de persuasion des technologies qui les utilisent :

- fiabilité ;
- expertise ;
- crédibilité présumée (*i.e.* crédibilité associée aux technologies par les individus) ;
- crédibilité de surface (*i.e.* crédibilité de la technologie après évaluation initiale) ;
- réputation (*i.e.* approbation des tiers) ;

Rôle d'outil
Réduction
Tunneling
Adaptation
Suggestion
Auto-surveillance
Surveillance
Conditionnement
Rôle de média
Cause et effet
Répétition virtuelle
Récompense virtuelle
Simulation dans un contexte réel
Rôle social
Attractivité
Similarité
Louange
Réciprocité
Autorité

TABLE 2.1 – Principes de persuasion de Fogg [18]

- crédibilité confortée (*i.e.* crédibilité renforcée avec le temps) ;
- perfection (*i.e.* crédibilité si le système ne commet jamais d'erreur).

Un résumé détaillé de ces principes est consultable dans le travail de recherche effectué par Foulonneau, Calvary et Vilain [19].

En 2009, Oinas-Kukkonen et Harjumaa [50] effectuent des recherches plus poussées dans l'application des TP. Ils mettent en avant une faiblesse du modèle de Fogg en précisant que ce modèle n'explique pas comment transformer les principes de conception suggérés par celui-ci en exigences fonctionnelles mises en oeuvre dans un système réel. Ils proposent donc un modèle de développement pour des solutions logicielles persuasives, appelé *Persuasive System Design* (PSD). En se basant sur le résultat de leurs recherches, ils démontrent qu'un système persuasif se développe en trois étapes (FIGURE 2.4).

Dans la première étape, ils définissent sept postulats nécessaires à la conception d'un système persuasif.

1. Les technologies de l'information ne sont jamais neutres, influençant d'une manière ou d'une autre les attitudes et le comportement des gens ;
2. Les gens aiment que leurs points de vue sur le monde soient organisés et cohérents ;
3. La persuasion est souvent progressive ;
4. Les itinéraires directs et indirects de changement de comportement sont des stratégies clés de la persuasion ;

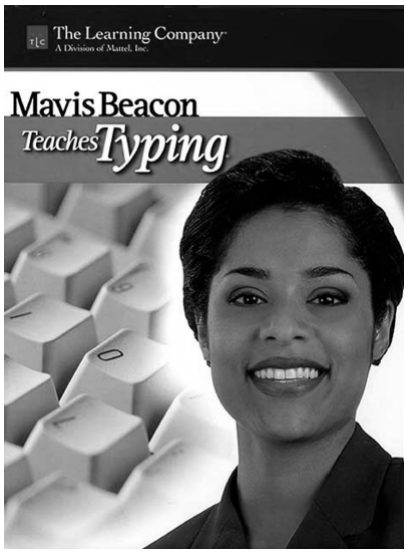

FIGURE 2.3 – Exemple d'autorité fictive dans le cadre de la persuasion à travers un rôle social [18]

5. Les systèmes persuasifs doivent avoir pour objectif d'être à la fois utiles et faciles à utiliser ;
6. Les systèmes persuasifs devraient être discrets ;
7. La persuasion par le biais de systèmes de persuasion devrait toujours être transparente.

La deuxième étape expose le contexte de la persuasion (*i.e.* les éléments qui peuvent influencer la mise en place de celle-ci) en segmentant les différents processus y afférents :

1. **L'intention** est composée de deux éléments. D'un côté le *persuadeur* (*i.e.* le concepteur du système) qui développe le système dans le but de persuader les utilisateurs et de l'autre le *comportement cible* que l'utilisateur doit atteindre grâce à ce système.
2. **L'événement** est composé de plusieurs sous-éléments. Le contexte d'*utilisation* qui concerne les fonctionnalités et les caractéristiques du domaine. Le contexte *utilisateur* qui regroupe les traits de caractère de l'utilisateur (*e.g.* objectif, motivation, style de vie, etc.) pour cerner au maximum la personnalité et enfin le contexte *technologique* qui concerne les fonctionnalités dépendant de la technologie.
3. **La stratégie** est définie par deux éléments. Premièrement le *message* qui concerne la forme et le contenu utilisés par le persuadeur pour transmettre les éléments persuasifs en vue d'atteindre un changement de comportement ou d'attitude. Deuxièmement la *route* qui peut être directe si on utilise une approche basée sur un nombre limité d'arguments solides ou indirecte si cette approche repose sur un certain nombre de faits. Ces deux approches ne sont pas exclusives et peuvent donc être combinées selon les besoins.

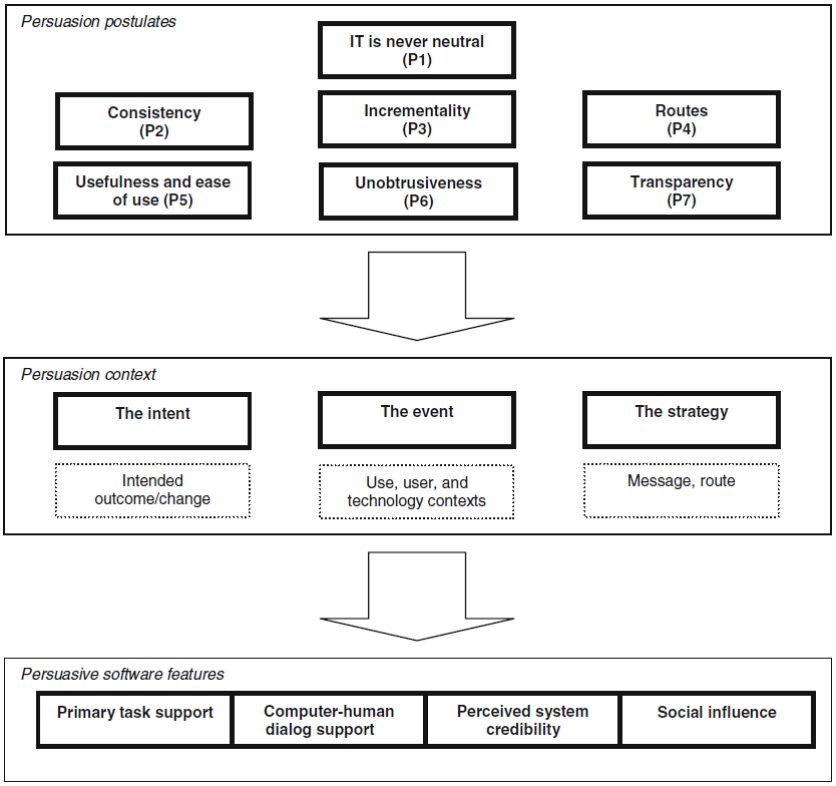

FIGURE 2.4 – Étapes du développement d'un système persuasif [50][48]

La dernière étape consiste à analyser et à sélectionner les principes persuasifs adéquats afin de les intégrer dans le processus de spécification des exigences. Ce processus constitue d'ailleurs l'une des étapes les plus importantes dans la conception d'un logiciel. La réussite de l'intégration de TP dans un système est donc intrinsèquement liée à l'aspect qualitatif et à la crédibilité de celui-ci.

Pour compléter ce modèle, nous pouvons également citer le travail de synthèse effectué par Brangier et Nemery [7] sur un ensemble d'articles dédiés au design persuasif. Cette synthèse fait ressortir une ligne directrice des critères à utiliser dans la conception des interfaces basées sur les TP. Le résultat de cette recherche met en avant huit critères regroupés dans deux différentes sections : les critères *statiques* et les critères *dynamiques* (FIGURE 2.5 p.16). Les critères statiques agissent sur les éléments graphiques ou textuels (*i.e.* le contenu) alors que les critères dynamiques opèrent sur l'organisation chronologique de l'interaction [8]. Ces critères rassemblent les composants techniques à mettre en place dans le cadre du développement d'une TP.

Les critères que nous venons de voir sont à associer à la prise en compte du comportement cible car pour pouvoir modifier un comportement, il convient de comprendre comment fonctionne la psychologie humaine et plus spécifiquement les facteurs qui guident ce comportement. Fogg [15] propose un modèle permettant d'atteindre un comportement cible (FIGURE 2.6 p.17).

D'après ce modèle, le comportement cible ne se déclenchera que s'il réunit trois conditions : *motivation*, *aptitude* et *déclencheur*. Ce modèle nous explique que plus la

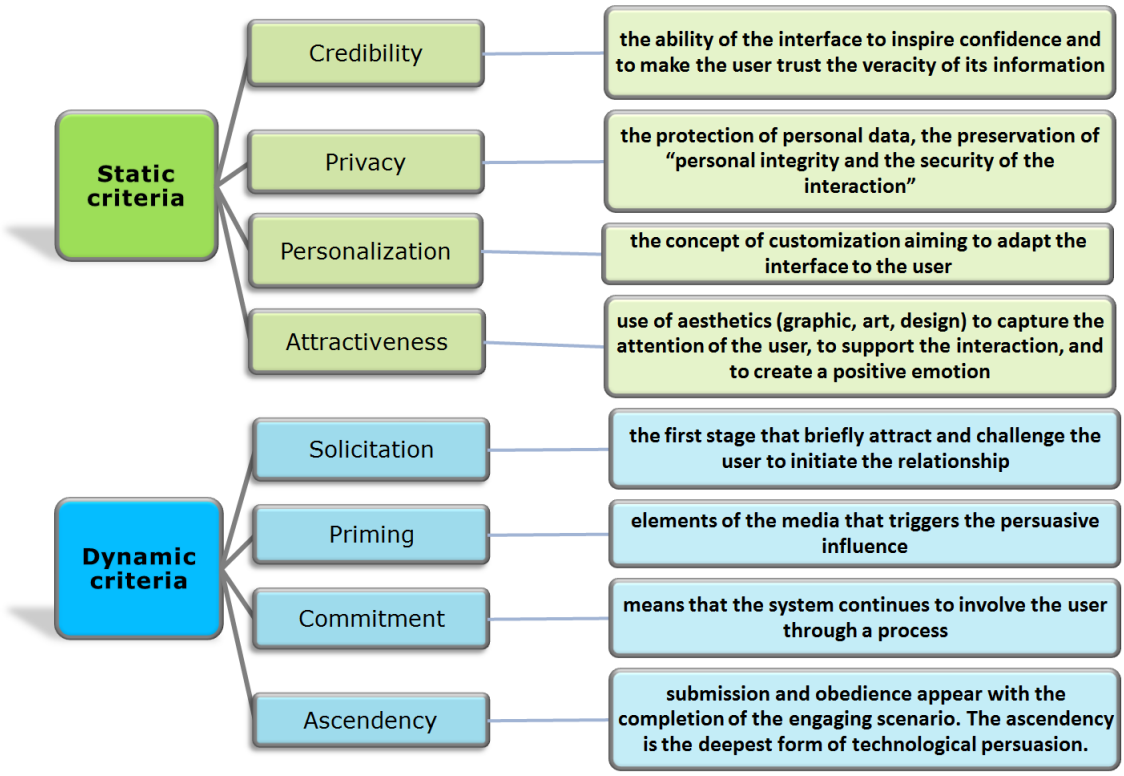

FIGURE 2.5 – Architecture générale des huit critères d'interaction persuasive [7]

motivation et la capacité à réaliser une tâche sont élevées et le déclencheur opportun, plus la probabilité d'atteindre le comportement cible augmente. C'est sur base de ce constat que repose le concept de *tiny habits* qui propose de modifier durablement son comportement par petites touches successives afin de ne pas être submergé par l'ampleur d'une tâche trop importante.

Pour mieux évaluer la bonne approche, Fogg [16] a conjointement développé un assistant de comportement dont l'objectif est de faire correspondre différents types de comportements cibles avec des solutions permettant de les atteindre (FIGURE 2.7 p.17). Il existe en effet une variété de comportements cibles ainsi qu'une temporalité pour ceux-ci. Cette temporalité est exprimée de trois façons. Atteindre un comportement cible *une fois*, sur *une période de temps* ou alors de manière *définitive*. Les comportements cibles sont répartis en 5 catégories. Il peut s'agir d'un *nouveau comportement*, d'un *comportement familier*, d'*augmenter*, de *réduire* ou d'*arrêter* un comportement existant.

2.1.3 Synthèse de la recherche sur les technologies persuasives

D'après les résultats publiés dans [28], près de 50% des TP recensées concernent la santé, environ 20% concernent l'écologie, 10% concernent l'éducation, 6% concernent l'économie, 6% concernent la sécurité et enfin 2% concernent le divertissement (les 6% restant ne répertorient pas de domaine spécifique) (FIGURE 2.8 p.18).

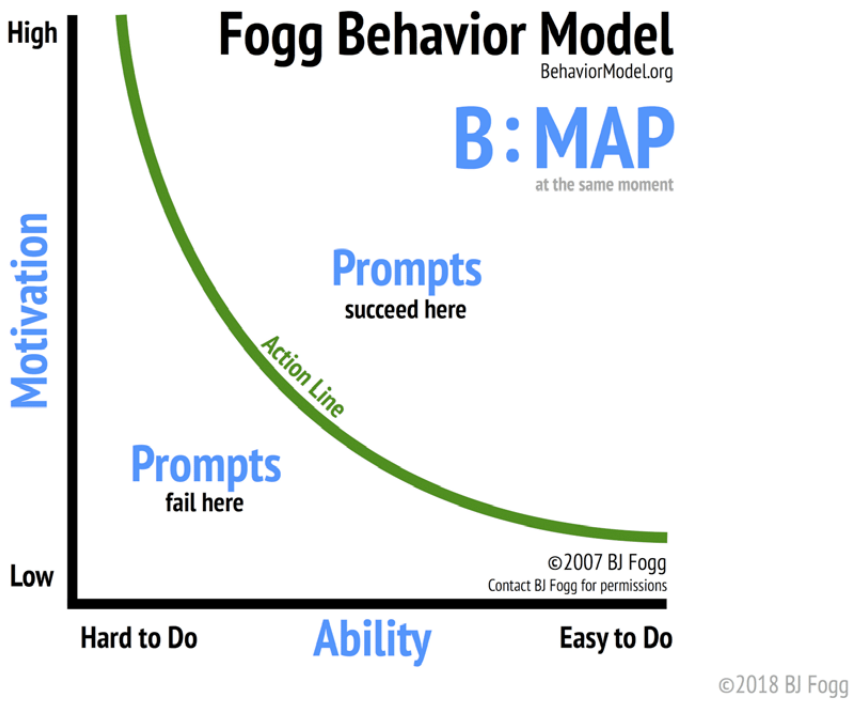

FIGURE 2.6 – Modèle comportemental de Fogg [17]

	Green behavior Do new behavior, one that is unfamiliar	**Blue** behavior Do familiar behavior	**Purple** behavior Increase behavior intensity or duration	**Gray** behavior Decrease behavior intensity or duration	**Black** behavior Stop doing a behavior
Dot behavior is done one-time	**GreenDot** Do new behavior one time *Install solar panels on house*	**BlueDot** Do familiar behavior one time *Tell a friend about eco-friendly soap*	**PurpleDot** Increase behavior one time *Plant more trees & local plants today*	**GrayDot** Decrease behavior one time *Buy fewer bottles of water now*	**BlackDot** Stop doing a behavior one time *Turn off space heater for tonight*
Span behavior has specific duration, such as 40 days	**GreenSpan** Do new behavior for a period of time *Carpool to work for three weeks*	**BlueSpan** Do familiar behavior for a period of time *Bike to work for two months*	**PurpleSpan** Increase behavior for a period of time *Take public bus for one month*	**GraySpan** Decrease behavior for a period of time *Take shorter showers this week*	**BlackSpan** Stop a behavior for a period of time *Don't water lawn during summer*
Path behavior is done from now on, a permanent change	**GreenPath** Do new behavior from now on *Start growing own vegetables*	**BluePath** Do familiar behavior from now on *Turn off lights when leaving room*	**PurplePath** Increase behavior from now on *Purchase more local produce*	**GrayPath** Decrease behavior from now on *Eat less meat from now on*	**BlackPath** Stop a behavior from now on *Never litter again*

FIGURE 2.7 – Assistant de comportement de Fogg [16]

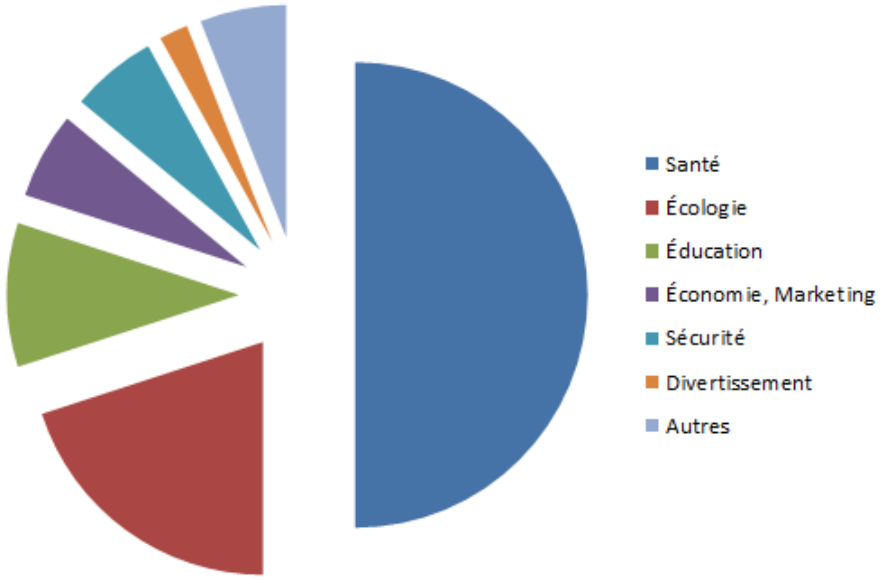

FIGURE 2.8 – Domaines des comportements cibles [28]

Le domaine des interfaces persuasives, des techniques de conception et des moyens de les mettre en oeuvre a déjà été largement couvert et des synthèses existantes peuvent être consultées afin d'avoir une vue d'ensemble sur le sujet [59][28][19].

Comme le démontrent ces différentes analyses, les systèmes persuasifs couvrent un large panel de domaines. Cependant, bien que l'implémentation de ces techniques fasse aujourd'hui l'objet de beaucoup d'engouement dans des secteurs tels que la santé, l'écologie, l'éducation ou encore les sites de vente en ligne, il reste des domaines peu exploités dans lesquels la mise en application de tels indicateurs pourrait avoir un effet bénéfique.

Dans le domaine du développement logiciel, l'utilisation de TP permettant une évolution comportementale qui vise à l'amélioration et à l'utilisation de bonnes pratiques de programmation serait évidemment pertinente. Judicieusement sélectionnées, les TP pourraient donc modifier le comportement et les habitudes des développeurs et permettraient conséquemment d'augmenter la qualité de conception d'un produit logiciel.

2.1.4 Émergence de la gamification

Afin de clarifier les termes et domaines de recherche existants, il nous apparaît pertinent de parler de l'émergence de la gamification. Ce concept est en plein essor depuis une dizaine d'années et s'intègre dans le contexte global de l'expérience utilisateur, au même titre que la persuasion dans les systèmes d'information.

La gamification et les TP sont devenues des éléments omniprésents dans le cadre de la recherche des IHM et des systèmes d'information [46]. Une des premières définitions de la gamification est « l'utilisation d'éléments de *game design* dans des contextes autres que le jeu »[10].

Comme la définition le suggère, ces éléments sont principalement issus du monde du jeu et consistent, par exemple, à donner des indications sur un niveau d'expérience éventuel, d'octroyer des récompenses voire même d'utiliser l'ordinateur en tant qu'acteur social en lui inculquant un rôle, ce qui renvoie directement aux fondements de la *captologie* développés par Fogg (TABLE 2.1 p.13). La gamification produit des résultats positifs mais ses effets dépendent intrinsèquement du contexte d'implémentation ainsi que des utilisateurs [29].

2.1.5 Gamification et technologies persuasives

Comme nous venons de l'évoquer, il existe des différences subtiles entre les TP et la gamification. Ces différences résident dans l'utilisation des principes persuasifs employés et leur finalité. Les TP se focalisent davantage sur la persuasion sociale et communicative ainsi que le changement d'attitude alors que la gamification semble plus s'orienter sur la stimulation de la motivation des utilisateurs à travers des principes ludiques [28].

2.2 La qualité logicielle

En génie logiciel, la qualité est l'un des processus permettant de vérifier et de valider les exigences. Cependant, résumer la QL en quelques lignes est une mission difficile tant la littérature sur le sujet est vaste et souvent même subjective. Nous nous baserons donc principalement sur le standard ISO pour définir et délimiter le champ d'action qui nous intéresse dans notre démarche.

2.2.1 Définition

Il existe un certain nombre de définitions pour caractériser la qualité d'un logiciel. Selon le *Guide to the Software Engineering Body of Knowledge* (SWEBOK) [26], la qualité d'un logiciel fait référence

- aux caractéristiques souhaitables des produits logiciels ;
- à la mesure dans laquelle un produit logiciel particulier possède ces caractéristiques ;
- aux processus, outils et techniques utilisés pour atteindre ces caractéristiques.

De nombreux modèles de qualité ont été proposés (*e.g.* McCall [39], Boehm [5], Dromey [12], etc.). Ces modèles regroupent les attributs internes et externes du système qu'il est primordial de surveiller afin d'en mesurer l'efficacité des composants. Le standard ISO 9126 [30] définit la *fonctionnalité*, la *fiabilité*, l'*utilisabilité*, l'*efficacité*, la *maintenabilité* et la *portabilité* comme étant les caractéristiques de qualité essentielles d'un produit logiciel.

Il existe plusieurs domaines de connaissances en QL (FIGURE 2.9) et nous nous intéresserons, dans le cadre de notre recherche, à celui se rapportant aux *Outils de Qualité Logicielle*. En effet, ces outils vont permettre de quantifier les attributs de qualité considérés.

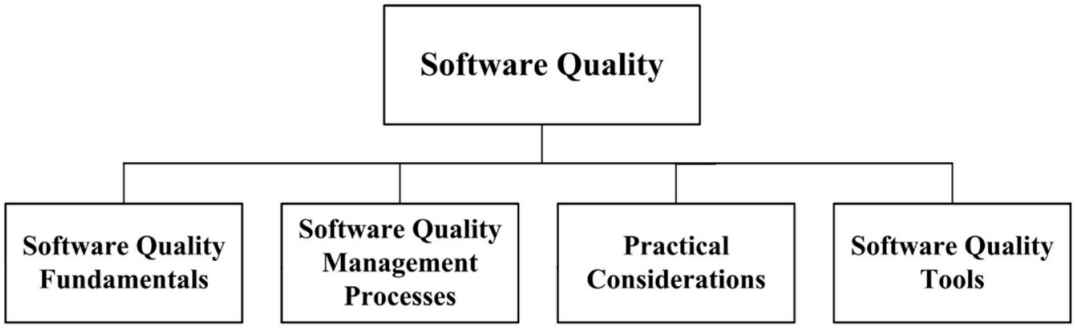

FIGURE 2.9 – Rubriques des domaines de connaissances en QL [26]

2.2.2 Les métriques logicielles

Dans un objectif de quantification de la qualité d'un logiciel, il convient d'utiliser des ML. Ces métriques vont permettre de mesurer la valeur d'une propriété technique ou fonctionnelle d'un système et vont déterminer quels sont les composants ayant une complexité élevée et pour lesquels les risques d'erreurs deviennent par conséquent plus grands. C'est une technique très utile pour contrôler le développement et la maintenance d'un produit. Nous nous basons sur les hypothèses suivantes concernant les ML [60] :

1. Une propriété logicielle peut être mesurée précisément ;
2. Il existe une relation entre ce que l'on peut mesurer et ce que l'on veut savoir car seuls les attributs internes sont mesurables alors que ce sont les attributs de qualité externes qui sont intéressants à mesurer car ils constituent les qualités désirables du code (*e.g.* maintenabilité)(FIGURE 2.10) ;
3. Il peut parfois être difficile de relier ce qui peut être mesuré aux attributs de qualité externes désirés.

La sélection des mesures à effectuer dépendra donc des caractéristiques de qualité sélectionnées. Les métriques dynamiques vont aider à mesurer l'efficacité et la fiabilité d'un programme alors que les métriques statiques se rapporteront plus à l'évaluation de la complexité, la compréhension et la maintenabilité du code. De plus, chaque indicateur repris dans les ML à surveiller doit faire l'objet d'une configuration personnalisée et les seuils sélectionnés le seront en fonction des besoins et des exigences de qualité retenus.

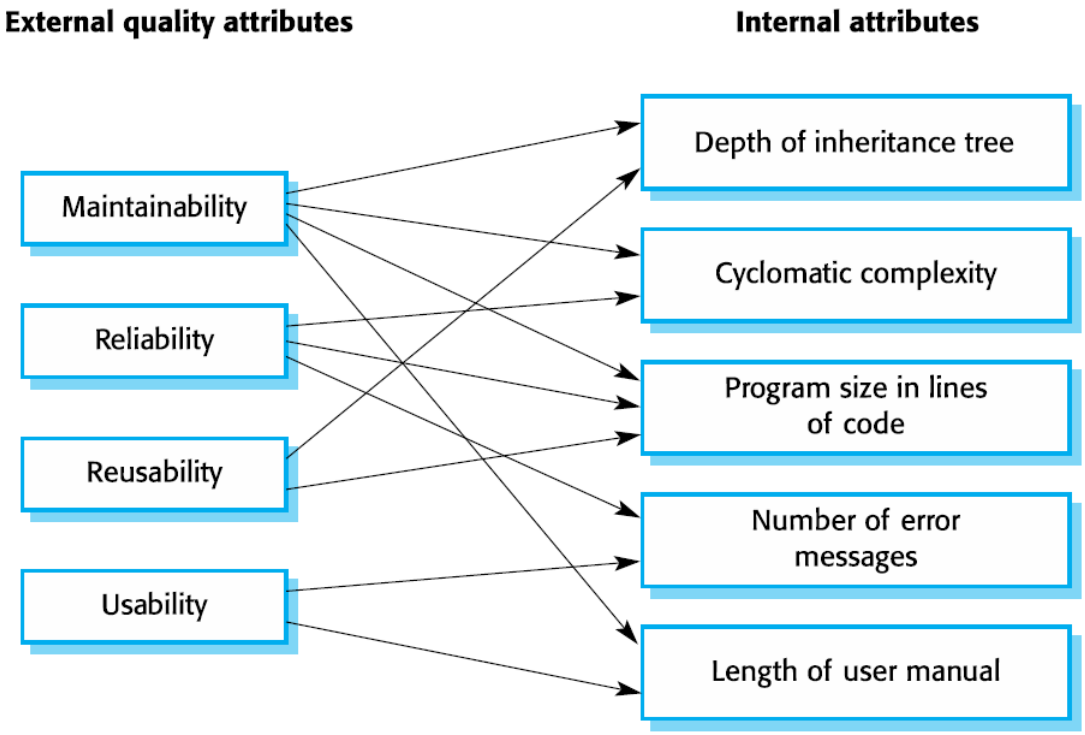

FIGURE 2.10 – Relation entre des attributs de qualité internes et externes [58]

Les métriques concernent le logiciel lui-même mais peuvent aussi se rapporter aux processus de développement et à la documentation existante. Les ML peuvent donc être classées en trois catégories [36] :

1. Métriques de processus ;
2. Métriques de produit ;
3. Métriques de ressource.

Comme l'objectif est d'optimiser la qualité du code, nous nous focaliserons sur les métriques de produit. Elles vont nous aider à analyser le code par l'évaluation de différents critères configurés afin de vérifier si le produit est conforme aux exigences de qualité souhaitées. Elles peuvent être dynamiques si elles sont collectées à l'exécution du programme ou statiques si elles sont collectées sur base d'une analyse du code.

L'intérêt de parler de ces indicateurs réside dans le souhait d'étudier le comment d'une implémentation efficace des TP en rapport avec la QL. Pour ce faire, il convient de préciser par l'entremise de quels moyens cela est réalisable. En effet, si les TP ont déjà fait leurs preuves dans un panel de domaines divers et variés, leur application dans un IDE doit encore être clarifiée.

2.2.3 Refactorisation

Il existe différentes méthodes pour augmenter la qualité du code produit par un développeur mais il est difficile de parler de QL sans évoquer la refactorisation. Celle-ci est définie par Fowler [20] comme le changement apporté à la structure interne

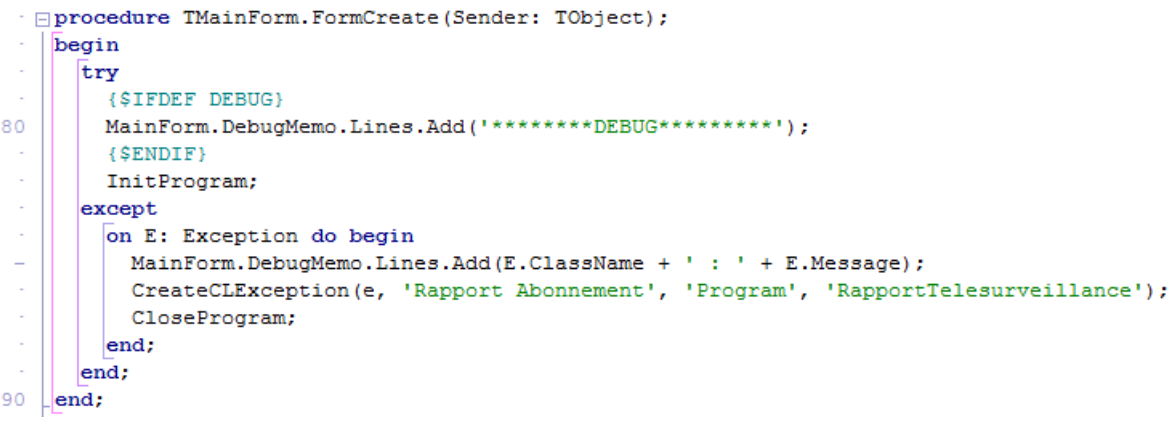

FIGURE 2.11 – Mise en valeur visuelle des mots-clés et de l'indentation du code dans RAD Studio 10.3

d'un logiciel pour le rendre plus facile à comprendre et moins coûteux à modifier sans changer son comportement observable.

La refactorisation est donc un procédé inévitable si l'on souhaite préserver les qualités d'un logiciel. De fait, le cycle de vie d'un logiciel implique de nombreux changements, qu'ils soient fonctionnels ou non fonctionnels. Chaque modification apportée au code, souvent sans une connaissance approfondie de l'architecture globale du projet, implique qu'au fil du temps, un logiciel aura tendance à se dégrader. Sans entrer en détail sur le sujet, nous pouvons distinguer deux types de refactorisation qui ne sont pas exclusifs. D'une part la refactorisation continue qui est intégrée dans le processus de développement d'un logiciel et d'autre part, la refactorisation a posteriori d'un projet (souvent ancien) qui s'inscrit alors dans un processus de réingénierie. Quelle que soit sa nature, la refactorisation a pour objectif d'améliorer la QL en rendant le code plus lisible et en réduisant les bugs potentiels.

De nombreux outils existent pour aider les développeurs dans cette tâche, qu'ils soient génériques ou développés pour un langage de programmation spécifique. Certains IDE embarquent même nativement ces outils qui peuvent être utilisés pour l'amélioration de la qualité du code. Ces outils ne concernent pas toujours exclusivement la détection de métriques potentiellement toxiques et l'analyse statique et dynamique du code n'est pas le seul levier sur lequel s'appuyer pour augmenter la qualité du code. D'autres moyens peuvent également être utilisés pour apporter une plus-value dans l'optimisation qualitative d'un produit logiciel. Nous pouvons citer les outils d'aide à la refactorisation proposant de simplifier le travail du développeur dans cette tâche souvent difficile et fastidieuse (*e.g.* extraire une méthode ou une interface, remonter un membre d'une classe, déclarer une variable, etc.). De plus, les IDE modernes formatent souvent le code à l'aide d'éléments visuellement distinctifs destinés à mettre en valeur les informations importantes du code. L'affichage ainsi optimisé peut permettre, de manière souvent attractive et intuitive, de sensibiliser le développeur à l'application de bonnes pratiques de développement (*e.g* Mots-clés d'un langage de programmation, Affichage visuel pour respecter une indentation propre et aérée du code) (FIGURE 2.11).

Notre but n'est pas ici de faire un inventaire exhaustif de l'ensemble des outils et techniques existants mais plutôt d'observer le contenu d'une sélection d'entre eux afin d'analyser la manière dont ils traitent les informations et surtout comment les résultats obtenus sont mis à disposition des développeurs. Il faut en effet pouvoir intuitivement profiter de la puissance de ces outils et de leur résultat sans se sentir submergé par l'ampleur de la tâche que cela peut représenter. Le volume d'informations à traiter pouvant généralement être conséquent, eu égard à l'importance de certains projets.

C'est une des possibilités offertes par les TP qui, dans ce contexte, peuvent apporter les moyens nécessaires pour aider le développeur (*i.e.* l'utilisateur) à optimiser sa manière d'utiliser ces outils. Ceci dans le but de modifier son comportement et ses habitudes de programmation par la prise en considération naturelle et intuitive des injonctions proposées par un IDE via ces mêmes outils.

3. Méthodologie

3.1 Motivations

Notre recherche est motivée par le besoin d'établir la connexion entre les TP et la QL dans un IDE. Elle porte donc sur l'exploration des améliorations que peuvent apporter les systèmes persuasifs dans le cadre du développement logiciel et de l'application de bonnes pratiques de programmation. Selon le comportement souhaité ou l'objectif à atteindre, il devrait exister un moyen de motiver un développeur à atteindre celui-ci grâce à l'utilisation opportune de TP. Cela nous amène à poser notre questionnement :

Comment utiliser les TP pour améliorer la qualité du développement logiciel ?

Pour tenter de répondre au mieux à cette question, nous devons en premier lieu établir par quel lien les TP vont pouvoir influencer la QL. Pour ce faire, nous devons nous demander quels sont les outils ou techniques les plus recommandés pour y parvenir.

L'absence d'un état de l'art concernant ce domaine de recherche nous a poussé à établir une revue de la littérature pour tenter de résumer la situation actuelle et pour identifier des pistes de recherches possibles. Nous avons suivi pour ce faire les lignes directrices reprises dans [35].

Quels sont les outils et techniques existants qui permettent d'optimiser la QL dans un IDE ?

Nous avons introduit la technique du relevé des ML dans la présentation de la QL car il est primordial lors d'un développement de mesurer les attributs de qualité d'un logiciel. Cette technique est réalisable grâce à la présence d'outils intégrés, nativement ou pas, dans un IDE.

Nous avons montré que les métriques de produit sont intéressantes car elles recensent principalement des informations sur la qualité du code source (*e.g.* nombre de classes, nombre de lignes de code par méthode, etc.). Elles constituent donc un indicateur indispensable pour informer l'utilisateur sur la qualité de son travail.

Cependant, les ML brutes ne suffisent pas toujours à faire prendre conscience de l'existence d'un problème. Tout dépendra de l'expérience du développeur. Pour l'aider dans cette tâche, il existe une variété d'AC qui utilise ces métriques.

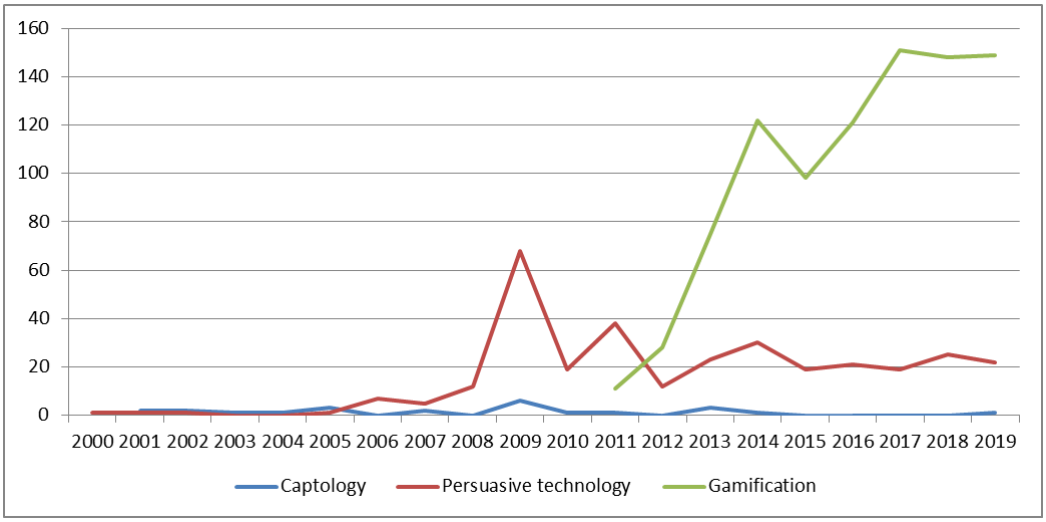

FIGURE 3.1 – Nombre de résultats de recherche, par année, pour les mots-clés concernés à partir des titres d'articles, des mots-clés et des résumés dans la base de données *ACM Digital Library*

Ces outils sont mis à disposition des développeurs et permettent d'automatiser l'inspection du code pour détecter la présence éventuelle de *code smell* (CS) sur base de métriques toxiques. Un CS peut être défini comme une caractéristique indiquant qu'un code peut contenir un problème de conception. Précisons qu'il n'existe pas une liste exhaustive de CS et que leur nature varie fortement (certains sont même idiomatiques). 24 CS ont été répertoriés dans l'ouvrage [20].

En outre, nous avons décidé d'intégrer dans notre recherche les systèmes de recommandation (SR) car ces outils peuvent également permettre d'optimiser la qualité du développement logiciel. C'est grâce à ces SR qu'une information pertinente peut être remontée de manière personnalisée ou, à tout le moins, sélectionnée selon des critères prédéfinis. Les SR ont émergé suite à un besoin naissant relatif à la masse conséquente d'informations à traiter. Leur but étant d'offrir la bonne information, au bon moment [47].

Le chaînon manquant est d'utiliser des TP pour ce type d'outils afin d'améliorer la qualité du code [53]. Nous verrons que leur utilisation efficace permet indéniablement d'augmenter cette qualité mais ces outils ne disposent malheureusement pas toujours d'une mise en lumière suffisante pour être exploités à bon escient, quand ils ne sont pas tout simplement ignorés ou inconnus [22].

Nous avons évoqué l'émergence de la gamification dans la présentation des TP. C'est plus précisément dans les années 2010 que ce terme prend son essor. Conjointement aux TP, la gamification permet d'orienter l'utilisateur vers un résultat escompté par l'entremise d'éléments spécifiques destinés à le motiver. En comparaison avec les articles concernant les TP et la *captologie*, la littérature concernant la gamification a rapidement augmenté depuis 2011 comme le démontre le recensement que nous avons effectué sur les articles publiés dans la base de données *ACM Digital Library* entre 2000 et 2019 (FIGURE 3.1). Puisque la gamification utilise des techniques persuasives, nous l'avons intégrée comme critère dans notre processus de sélection.

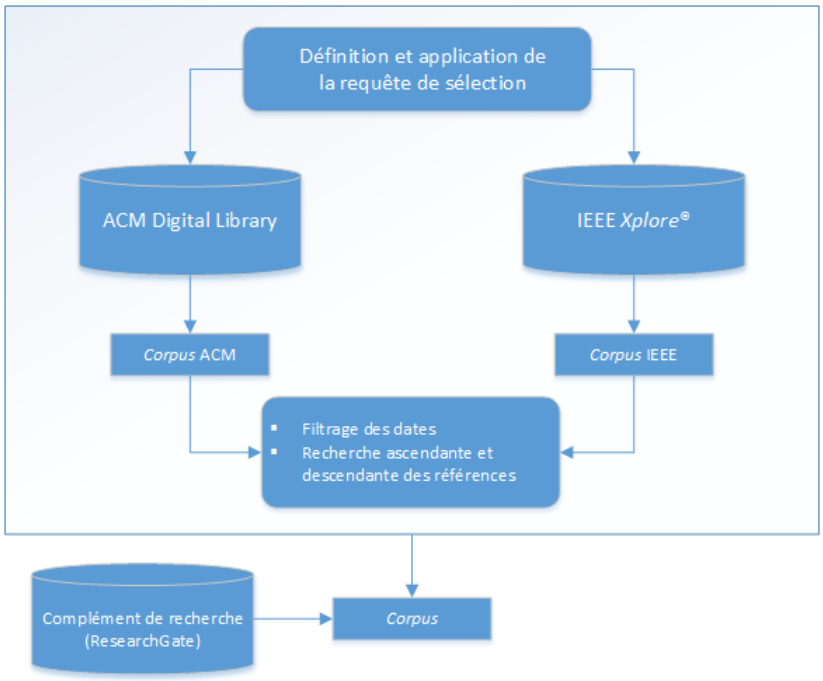

FIGURE 3.2 – Protocole de sélection

Nous pourrons dès lors vérifier quelles sont les différences entre la gamification et les TP en fonction des principes de conception utilisés.

3.2 Méthode de recherche

Pour mener à bien notre recherche, nous avons sélectionné les documents dont le contenu est au croisement des deux domaines présentés dans notre contexte (*i.e.* TP et QL) afin d'en extraire les principales caractéristiques et tendances, toujours dans une optique d'optimisation de la qualité du code des développements de solutions informatiques. En croisant les outils et technologies utilisés dans les IDE avec les facteurs de persuasion les plus pertinents pour ceux-ci, notre objectif est de dégager des pistes de recherche possibles dans le domaine.

Pour déterminer et isoler au mieux les TP en rapport avec les outils sur lesquels elles s'appliquent dans un IDE, la définition de nos requêtes de sélection a été établie sur deux axes. Le premier axe (a) se focalise sur les AC et les ML. Le second axe (b) concerne principalement l'analyse des SR embarqués dans les IDE.

Les bases de données *ACM Digital Library* et *IEEE Xplore* ® ont été retenues car elles constituent les moteurs de recherche principaux dans le domaine de l'informatique [34].

Les recherches ont été effectuées selon la méthodologie suivante :

1. Les TP ayant seulement émergé à la fin des années nonante, nous décidons de recenser les documents des deux dernières décennies, soit ceux dont les dates

de publication sont comprises entre le 1er janvier 2000 et le 31 décembre 2019 inclus. Nous décidons d'ajouter ce premier critère d'inclusion directement dans notre requête de sélection initiale. Après avoir identifié un ensemble de mots-clés correspondants à notre champ de recherche, nous exécutons les requêtes suivantes sur la base de données *ACM Digital Library* :

(a) *[[Publication Title : gamification] OR [Publication Title : persua*]] AND [[Publication Title : ide] OR [Publication Title : software] OR [Publication Title : developpement] OR [Publication Title : "code quality"] OR [Publication Title : tool]] AND [[Abstract : gamification] OR [Abstract : persua*]] AND [Publication Date : (01/01/2000 TO 12/31/2019)]*. Nous obtenons 42 résultats.

(b) *[[Publication Title : recommender] OR [Publication Title : recommendation]] AND [[Abstract : ide] OR [Abstract : developpment]] AND [Publication Date : (01/01/2000 TO 12/31/2019)]*. Nous obtenons 21 résultats.

2. Nous procédons ensuite à une analyse des titres, mots-clés et résumés afin d'extraire les publications pertinentes en rapport avec les outils d'aide aux bonnes pratiques du développement logiciel.

   Les critères d'inclusion suivants ont été appliqués pour la sélection des documents :

   - recherche empirique et expérimentale des TP dans le cadre du développement logiciel ;
   - prototypage d'outil intégré dans un IDE ;
   - article sur l'étude des relations possibles entre TP, gamification et QL.

   Les critères d'exclusion suivants ont été utilisés :

   - QL en rapport avec le management de la qualité ;
   - articles en lien avec ITIL.

   Nous retenons 13 articles.

3. Sur base de cette première sélection, nous alimentons notre corpus en effectuant une recherche ascendante et descendante des citations afin d'extraire de nouveaux documents en rapport avec le sujet traité en s'inspirant des stratégies proposées dans [42]. Nous obtenons 10 documents supplémentaires.

4. Nous reproduisons les étapes 1 à 3, sur la base de données *IEEE Xplore* ®, avec les requêtes suivantes :

   (a) *("Document Title" :gamification OR "Document Title" :gamified OR "Document Title" :persuasive OR "Document Title" :persuasion) AND ("Document Title" :IDE OR "Document Title" :developpement OR "Document Title" :software OR "Document Title" :code quality OR "Document Title" :tool) AND ("Abstract" :gamification OR "Abstract" :persuasive OR "Abstract" :persuasion).*
   Nous obtenons 55 résultats.

   (b) *("Document Title" :recommender OR "Document Title" :recommendation) AND ("Abstract" :ide OR "Abstract" :developpment OR "Abstract" : persuasion).*
   Nous obtenons 19 résultats.

Après lecture et conformément à l'étape 2, nous décidons de garder 5 documents. Conformément à l'étape 3, nous procédons à l'ajout de 2 documents supplémentaires

5. Nous enrichissons notre corpus par une recherche complémentaire effectuée sur le moteur de recherche *ResearchGate*. Les critères de recherche, d'inclusion et d'exclusion sont les mêmes que pour les recherches effectuées sur la base de données *ACM Digital Library* et *IEEE Xplore* ®.

   Nous ajoutons 5 documents.

6. Notre corpus final se compose de 35 documents.

3.3 Structure de l'analyse

Nous avons décidé de mettre en relation certains indicateurs de notre corpus en suivant un processus inspiré par les lignes directrices de l'article [62]. Nous avons listé 1) les publications afin de mettre en relation 2) les TP appliquées, 3) les ML utilisées, 4) le type de "support" présenté par le document, 5) le type de système (AC ou SR), 6) l'IDE et les outils utilisés et enfin, 7) la taille du groupe de test et soit la durée de l'expérience, soit le recours à une étude qualitative (EQ) (Table 4.1 p.40).

3.3.1 Définition des principes de conception

Pour nous aider dans l'analyse de notre corpus, nous nous sommes basés sur le PSD [50] pour détecter les principes de conception les plus utilisés dans le cadre des TP. Ces principes ont fait l'objet d'un recensement et sont répartis dans 4 catégories selon leur nature (Table 3.1).

Sur base de notre analyse, nous avons pu identifier les principes les plus pertinents selon les outils et techniques dans lesquels ils sont utilisés. Bien que certains principes se chevauchent, il existe des différences significatives entre les SR et les AC (*e.g.* détection de CS). Cette différence se justifie par le fait que l'objectif de l'outil et son mode de fonctionnement impliquent que chacune de ces technologies s'appuie sur des techniques persuasives différentes, bien que non exclusives.

3.3.2 Recensement des métriques

Les ML suivantes ont été identifiées afin d'être mises en relation avec les principes de conception retenus. Nombre de classes (NC), nombre de références de méthode d'une autre classe dans une classe (NRMCC[1]), nombre d'instances d'une autre classe dans une classe (NICC[1]), nombre de méthodes (NM), nombre de méthodes dans une classe (NMC[1]), complexité cyclomatique (CC), nombre de lignes de code (LOC[1]), nombre de paramètres par méthode (NPM[1]), nombre d'attributs par méthode (NAM), profondeur d'héritage (PH), nombre de méthodes dont la première

1. ML pouvant être utilisées pour l'identification de CS de type *Feature Envy, Large Class, Long Method* ou encore *Data Clump*.

Tâche principale	
Principe	*Définition*
Réduction	Décomposition d'un comportement complexe en tâches simples pour aider l'utilisateur à atteindre son objectif
Tunneling	Utilisation d'un dispositif pour guider l'utilisateur à travers une série de process
Adaptation	Le pouvoir de persuasion du système sera plus grand s'il peut être adapté aux besoins, aux intérêts ou à la personnalité de l'utilisateur
Personnalisation	Un système qui fournit un contenu ou un service personnalisé à un plus grand pouvoir de persuasion
Auto-surveillance	Un système qui assure le suivi des performances ou du statut d'un utilisateur l'aide à atteindre ses objectifs.
Simulation	Un système est plus convaincant s'il peut fournir une simulation afin d'observer les causes et effets d'une action
Répétition	Un système qui permet de répéter un comportement peut opérer des changements d'attitudes dans le monde réel
Dialogue	
Principe	*Définition*
Louange	En faisant des éloges à l'utilisateur, le système lui permet d'être plus ouvert à la persuasion
Récompense	Un système qui récompense l'utilisateur peut avoir un grand pouvoir de persuasion
Rappel	Si le système rappelle à l'utilisateur le comportement cible à atteindre, celui-ci atteindra plus probablement son objectif
Suggestion	Un système qui offre des suggestions adaptées a un plus grand pouvoir de persuasion
Similarité	L'utilisateur sera plus facilement persuadé par des recommandations qui sont proches de sa manière de penser
Attractivité	Un système visuellement attractif sera plus persuasif
Rôle social	L'utilisateur utilisera plus probablement un système à des fins persuasives si celui-ci adopte un rôle social

Crédibilité du système	
Principe	*Définition*
Fiabilité	Un système disposera d'un plus grand pouvoir de persuasion s'il est considéré comme fiable
Expertise	Un système offrant une expertise à l'utilisateur aura un plus grand pouvoir de persuasion
Crédibilité	Le visuel d'un système et son ressenti doivent être de qualité
Sensation du monde réel	Un système qui met en valeur des personnes ou une organisation derrière son contenu ou ses services aura plus de crédibilité.
Autorité	Un système qui agit avec un rôle autoritaire aura un pouvoir de persuasion plus puissant
Approbation des pairs	Un système augmentera sa crédibilité et son pouvoir de persuasion s'il utilise les recommandations de tiers connus et appréciés
Vérifiabilité	Le système augmentera sa crédibilité et son pouvoir de persuasion s'il met à disposition de l'utilisateur des moyens de vérification externes de son contenu
Social	
Principe	*Définition*
Apprentissage social	L'observation de l'apprentissage d'un comportement d'autres personnes peut motiver un utilisateur à atteindre un objectif précis
Comparaison sociale	La motivation à atteindre un comportement cible peut être accrue si le système permet de comparer ses propres performances avec celles d'autres utilisateurs
Influence normative	Un système peut utiliser l'influence normative pour augmenter la probabilité qu'une personne adopte un comportement cible
Facilitation sociale	Un système qui permet de montrer que d'autres utilisateurs adoptent un comportement permettra à un utilisateur d'être plus enclin à atteindre le même comportement
Coopération	Les systèmes coopératifs peuvent motiver un utilisateur à adopter le même comportement ou la même attitude
Compétition	L'utilisateur peut être plus motivé à atteindre un objectif ou un comportement cible si le système propose un mode compétitif
Reconnaissance	La reconnaissance et l'approbation des pairs peuvent avoir un effet bénéfique sur la volonté d'atteindre un comportement cible

TABLE 3.1 – Catégories des principes de conception d'un système persuasif [50]

lettre est une majuscule (MMAJ), nombre de variables dont la première lettre est une minuscule (VMIN), nombre de dépendances entre les modules, aussi appelé couplage logique (NDM), nombre de commits (*i.e.* l'enregistrement effectif d'une transaction) effectués (NCO), nombre de commentaires (NCMT), nombre de parties de code dupliquées (CD), nombre de bugs identifiés (NBI) et enfin nombre de CS identifiés (NCSI). Ces deux dernières métriques (*i.e.* NBI et NCSI) sont un peu particulières car elles sont issues d'une détection manuelle et dépendent donc uniquement de l'utilisateur qui les a identifiées. Nous les avons malgré tout retenues car ce travail d'identification a été réalisé via un outil intégrant des TP.

3.3.3 Types de documents

Nous avons décidé de répartir les documents selon trois types :

- **Prototype** pour les documents traitant du développement et de l'expérimentation d'un outil dans un IDE ;
- **Framework** pour les documents proposant un cadre de développement théorique d'application de TP pour améliorer la QL ;
- **Étude** pour les documents théoriques analysant les TP dans le cadre du développement logiciel.

3.3.4 Système concerné

Comme notre recherche se concentre majoritairement sur les sytèmes de type AC ou SR, leur référencement permettra de faire émerger des tendances quant à l'utilisation de TP spécifiques.

3.3.5 IDE et outils éventuels

Nous avons recensé, le cas échéant, les IDE utilisés afin de pouvoir identifier les tendances concernant les langages les plus représentatifs. Les outils font référence soit à un prototype spécifiquement développé dans la publication concernée, soit à un outil existant dans lequel des TP ont été implémentées.

3.3.6 Groupe de test et durée

Nous avons intégré la taille des groupes de test en précisant le type de participants (*e.g.* étudiant, développeur novice). Nous avons également ajouté la durée de ces tests. Enfin nous avons précisé s'il s'agit d'une EQ.

4. Résultats

4.1 Analyse comparative

Afin de classer les documents et de répertorier le plus efficacement possible les points convergents, nous avons focalisé notre analyse sur un ensemble de principes de conception issus des TP. Nous avons cependant remarqué que certains documents retenus peuvent faire référence à ces principes sans que ceux-ci n'aient a priori été utilisés dans un processus volontaire d'intégration persuasive.

4.2 Constatations

La visualisation des résultats obtenus dans les deux technologies principalement retenues (*i.e.* AC et SR) montrent des tendances différentes (FIGURE 4.2, 4.4 p.34).

Concernant l'utilisation de TP ou de gamification dans un AC, le principe de récompense est utilisé dans plus de 70% des cas. 40% environ concernent le rôle social apporté par l'outil et l'intégration d'un mode compétitif. Environ 30% concernent les principes de similarité, d'attractivité et d'apprentissage social. Environ 20% concernent l'utilisation de la suggestion, de louange, d'adaptation, d'autosurveillance, de personnalisation, de fiabilité, d'expertise, de comparaison sociale et enfin d'influence normative. Environ 10% concernent les principes de rappel, de réduction, de répétition, de reconnaissance, de facilitation sociale, de crédibilité et d'autorité. Le reste des principes de conception est repris dans moins de 10% des cas.

Concernant les SR, le principe de conception le plus utilisé est la suggestion dans plus de 70% des cas. Environ 50% concernent le principe de similarité, de rôle social, d'apprentissage social et de comparaison sociale. 30 à 40% concernent l'adaptation, l'attractivité, la réduction, le rappel, la personnalisation, la répétition, l'influence normative et l'expertise. Environ 20% concernent la crédibilité, la fiabilité, la sensation du monde réel. Environ 10% concernent la reconnaissance, la coopération et l'autorité. Le reste correspond à moins de 10% des cas recensés.

Cette distinction entre les deux systèmes vient principalement de leur différence de finalité. Alors que le traitement effectué dans un AC est propice à l'implémentation de la gamification pour optimiser son efficacité (FIGURE 4.1 p.33), les conseils et informations donnés par un SR agiront plus sur l'aspect social, l'expertise et la suggestion pour renforcer son pouvoir de persuasion (FIGURE 4.3 p.34).

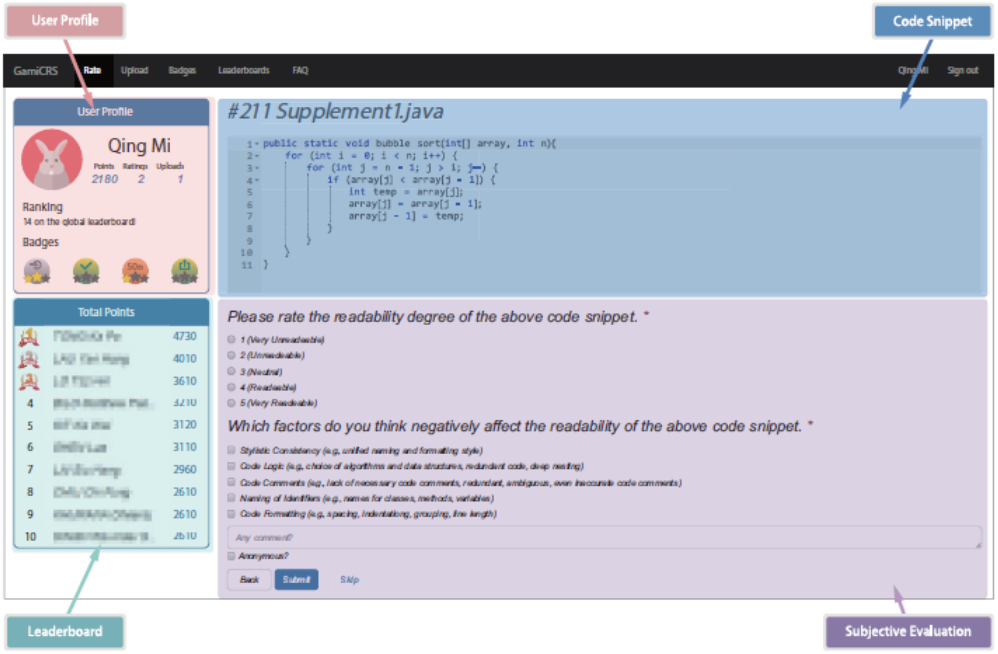

FIGURE 4.1 – Visualisation d'un outil pour l'analyse de CS [41]

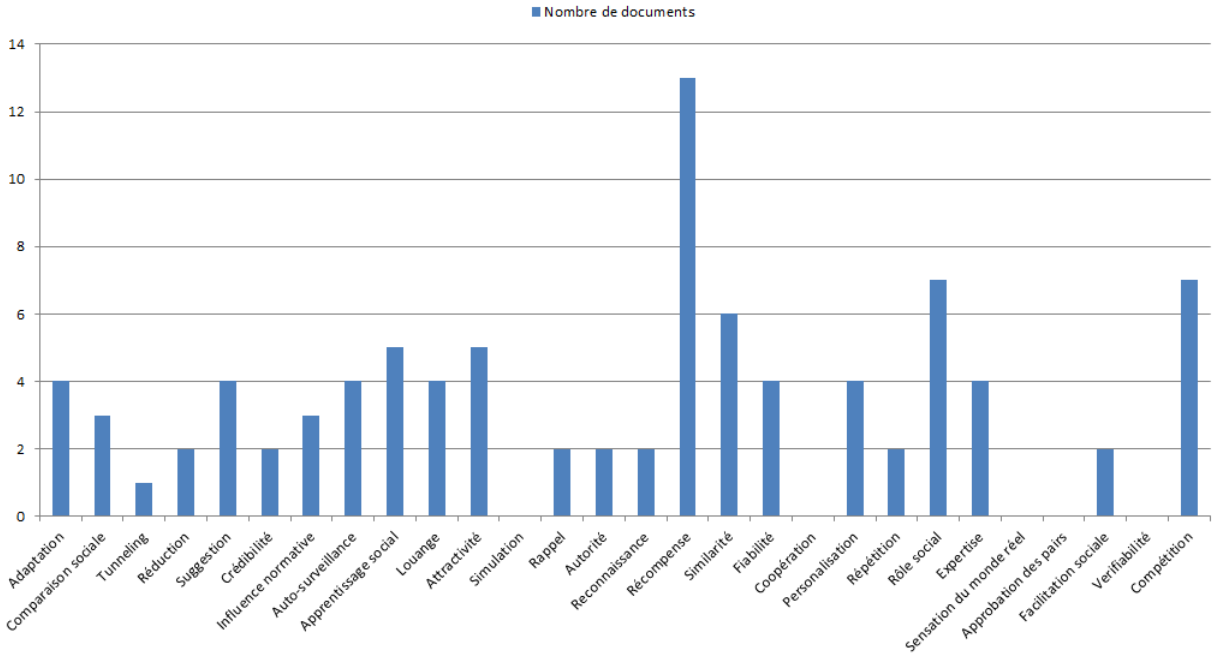

FIGURE 4.2 – Stratégies persuasives recensées dans les AC

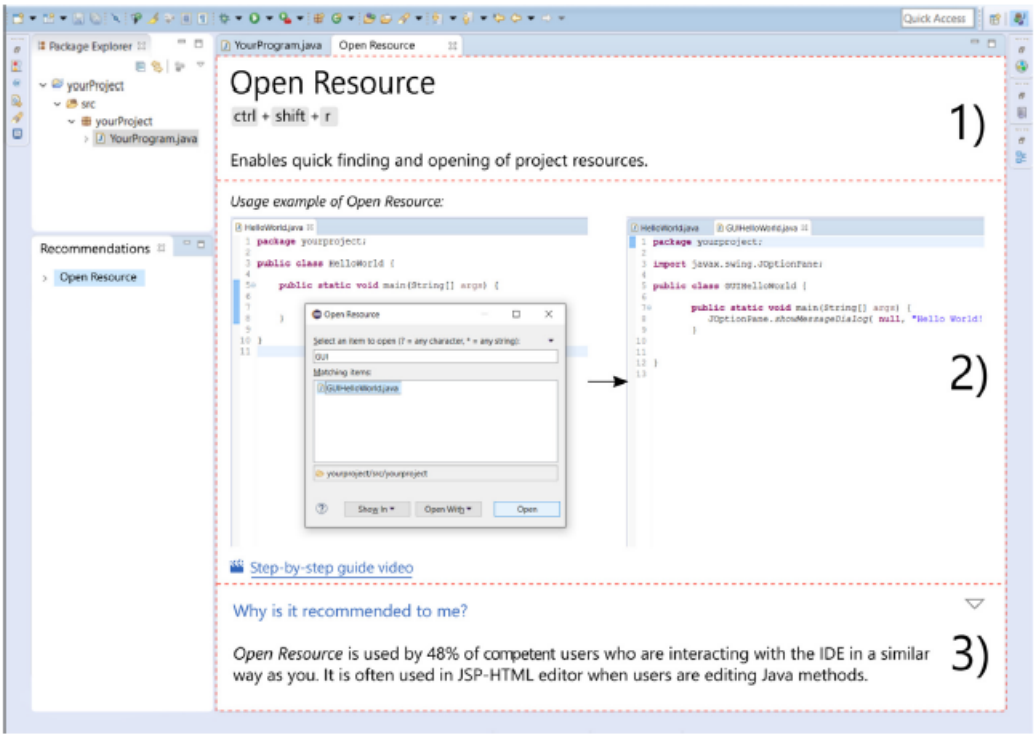

FIGURE 4.3 – Présentation des commandes d'un SR [24]

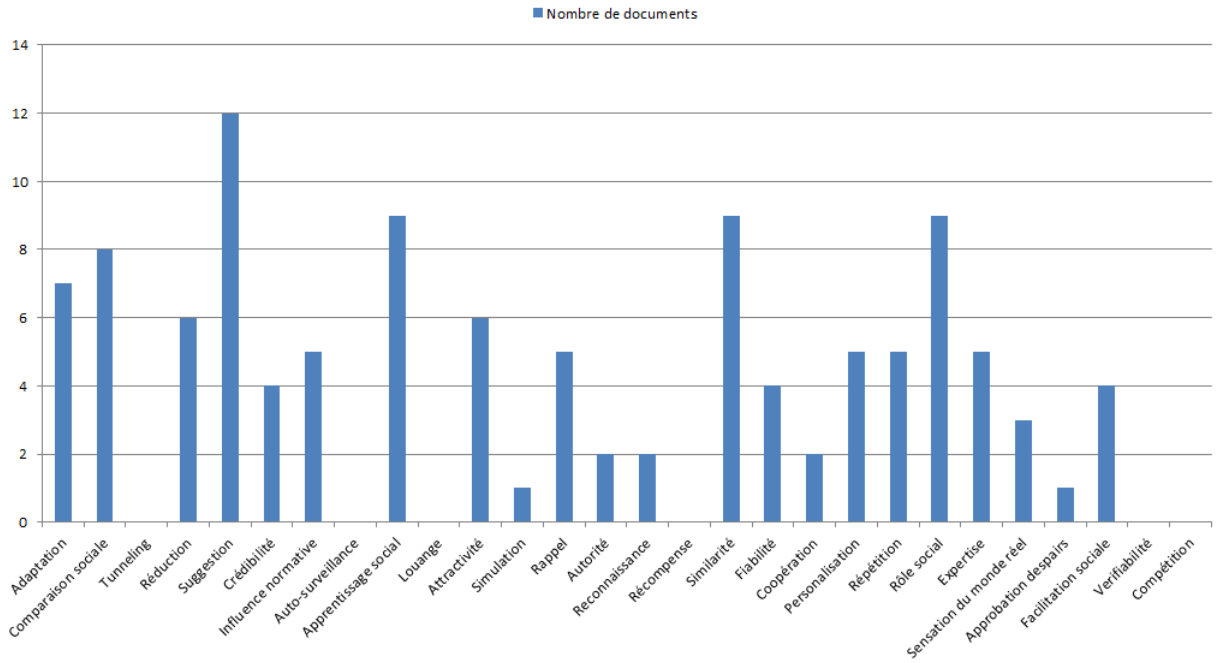

FIGURE 4.4 – Stratégies persuasives recensées dans les SR

Art.	TP	ML	Doc.	Sys.	IDE *Outil*	Groupe *Durée*
[14]	Récompense Similarité Compétition		Prototype	AC	Java (Eclipse) *CodeArena*	12 développeurs *1 mois*
[41]	Auto-surv. Apprent. Soc. Louange Récompense Fiabilité Répétition Rôle social Expertise Facilit. soc. Compétition		Prototype	AC	Java *GamiCRS*	161 étudiants (Soft. Design)
[3]	Apprent. soc. Compétition	LOC	Prototype	AC	Java *GBCatcher* *FindBugs*	6 étudiants (Comp. science)
[24]	Adaptation Comp. soc. Réduction Suggestion Apprent. soc. Similarité Personalis. Répétition Rôle social Expertise		Prototype	SR	Java (Eclipse) *Spyglass*	36 participants *EQ*
[43]	Suggestion Attractivité Rappel Récompense Répétition	NRMCC LOC NPM	Prototype	AC	Java (Eclipse) *StenchBlossom*	12 participants (6 dev. Java + 6 étudiants)
[61]	Rappel		Prototype	SR	Java (Eclipse) *Spyglass*	9 étudiants (Soft. Engin.) *1 mois*
[56]	Adaptation Comp. soc. Suggestion Apprent. soc. Récompense Similarité Rôle social Compétition	NICC LOC NRMCC	Prototype		Github	18 participants
[4]	Récompense	CD	Prototype	AC	Java	

Art.	TP	ML	Doc.	Sys.	IDE *Outil*	Groupe *Durée*
[63]	Adaptation Attractivité Personalis.	LOC NC NM NAM	Prototype	AC	Java C++ Smalltalk *CodeCity*	
[37]	Attractivité	LOC NC NM NDM	Prototype	AC	Java C++ Smalltalk COBOL *CodeCrawler*	
[21]	Comp. soc. Suggestion Apprent. soc. Personalis. Rôle social		Étude	SR	Java (Eclipse)	8 participants *10 semaines*
[38]	Adaptation Suggestion Attractivité Rappel Reconnaissance Similarité Personalis. Répétition		Étude			36 participants (25 h, 11 f) *10 semaines*
[32]	Adaptation Crédibilité Apprent. soc. Louange Autorité Similarité Fiabilité Personalis. Rôle social	LOC NCO	Prototype		Java *FindBugs* *Jenkins*	
[1]	Récompense Rôle social Compétition		Étude			50 étudiants *6 semaines*
[52]	Comp. soc. Réduction Infl. norm. Auto-surv. Récompense Similarité Rôle social Expertise Compétition	LOC	Étude			2 Teams *2 mois*

Art.	TP	ML	Doc.	Sys.	IDE *Outil*	Groupe *Durée*
[11]	Louange Récompense Rôle social Facilit. soc.		Étude			
[51]	Suggestion		Étude		Java (Eclipse) *SeeHawk*	
[64]	Réduction Facilit. soc.		Étude		Java (Android)	
[23]	Adaptation Apprent. Soc. Fiabilité Sens. monde réel		Prototype		Java *CoRe* *Vignelli* *SpyGlass*	19 étudiants (Programing) *6 semaines*
[33]	Auto-surv. Attractivité Récompense	NCMT NBI NCSI	Étude		*Bitbucket* *Phabricator* *Github CC* *Codebrage* *GamifiedSD*	183 étudiants
[22]	Adaptation Réduction Suggestion Infl. norm. Attractivité Similarité Répétition Expertise Sens. monde réel		Étude	SR	Java (Eclipse) *CoDis* *CNTX*	148 étudiants *40 semaines*
[44]	Adaptation Comp. soc. Réduction Suggestion Apprent. Soc. Coopération Rôle social Expertise Approb. pairs Facilit. soc.		Étude	SR	Java (Eclipse)	13 participants (4 experts et 9 novices)

Art.	TP	ML	Doc.	Sys.	IDE ***Outil***	Groupe ***Durée***
[9]	Adaptation Comp. soc. Réduction Suggestion Crédibilité Attractivité Similarité Fiabilité Personalis. Rôle social		Étude	SR	*ContentWise*	210 participants *EQ*
[55]	Simulation Rappel Similarité Répétition	CD	Étude	SR	*Quickfix*	
[2]	Auto-surv. Attractivité Récompense Expertise		Framework			
[54]	Adaptation Comp. soc. Réduction Suggestion Crédibilité Infl. norm. Apprent. soc. Attractivité Reconnaissance Similarité Coopération Personalis. Rôle social		Étude	SR		
[57]	Suggestion Rappel Similarité Sens. monde réel		Étude	SR	Java *Junit* *JHotDraw* *Jmove* *Jdeodorant*	

Art.	TP	ML	Doc.	Sys.	IDE *Outil*	Groupe *Durée*
[31]	Comp. soc. Suggestion Crédibilité Infl. norm. Apprent. soc. Attractivité Rappel Autorité Fiabilité Répétition Rôle social Facilit. soc.		Étude	SR		275 étudiants *EQ*
[25]	Comp. soc. Suggestion Crédibilité Infl. norm. Apprent. soc. Attractivité Autorité Similarité Rôle social Expertise		Étude	SR		148 participants *EQ*
[49]	Suggestion Crédibilité Infl. norm. Attractivité Rappel Autorité Reconnaissance Récompense Similarité Fiabilité Personalis. Rôle social Expertise		Framework			

Art.	TP	ML	Doc.	Sys.	IDE *Outil*	Groupe *Durée*
[53]	Tunneling Réduction Suggestion Infl. norm. Auto-surv.	NC NICC NM NMC CC LOC NPM PH MMAJ VMIN	Prototype	AC	C# (VS2010)	6 développeurs *6 semaines*
[47]	Infl. norm. Apprent. soc. Similarité Fiabilité Rôle social Facilit. soc.		Étude	SR		943 profils
[13]	Comp. soc. Apprent. soc. Récompense Compétition	LOC CD NCMT	Prototype	AC	C# (VS) Java (Android) *Sonar* *JavaDoc*	32 groupes (2 à 3 étudiants)
[27]	Adaptation Louange Reconnaissance Récompense Similarité Personalis.		Étude			
[45]	Comp. soc. Suggestion Apprent. soc. Rôle social Expertise		Étude	SR		18 étudiants 18 développeurs *EQ*

TABLE 4.1 – Analyse comparative du corpus

Au niveau du développement des prototypes d'AC, c'est le langage Java (majoritairement dans l'environnement *Eclipse*) qui est utilisé dans 10 cas sur les 11 analysés. C# (Visual Studio) est utilisé dans le dernier cas.

Nous constatons que sur les 22 groupes de test recensés, 11 impliquent des étudiants dans le domaine de l'informatique, 5 groupes impliquent des développeurs (4 groupes sans précision sur leur expérience et 1 groupe impliquant à la fois des experts et des novices).

L'analyse de notre corpus fait ressortir quatre documents [21][24][43][53] traitant spécifiquement de l'intégration de la persuasion dans un outil intégré à un IDE. Ils constituent donc une base utile pour aider à la mise en place de TP dans un IDE.

Lorsque nous observons la TABLE 4.2, nous apercevons un manque dans la recherche concernant les principes de conception tels que l'approbation des pairs, le sens du monde réel, la facilitation sociale ou encore la vérifiabilité. Ces concepts ne sont appliqués, dans le cas de notre corpus, ni dans les AC ni dans les SR. À l'opposé, les ML de type NC, NM, LOC, NCOM sont fortement utilisées, principalement dans les AC où elles sont couplées à des TP liées aux principes d'adaptation, de récompense et de compétition et au SR où elles sont couplées avec les principes de comparaison sociale, de suggestion, d'apprentissage social, d'attractivité et de rôle social.

	NC	NRMCC	NICC	NM	NMC	CC	LOC	NPM	NAM	PH	MMAJ	VMIN	NDM	NCO	NCMT	CD	NBI	NCSI
Adaptation	1	1	1	1			3		1					1				
Comp. soc.		1	1				3								1	1		
Tunneling	1		1	1	1	1	1	1		1	1	1						
Réduction	1		1	1	1	1	2	1		1	1	1						
Suggestion	1	2	2	1	1	1	3	2		1	1	1						
Crédibilité							1							1				
Infl. norm.	1		1	1	1	1	2	1		1	1	1						
Auto-surv.	1		1	1	1	1	2	1		1	1	1			1		1	1
Apprent. soc.		1	1				4							1	1	1		
Louange							1							1				
Attractivité	2	1		2			3	1	1				1		1		1	1
Simulation																1		
Rappel		1					1	1								1		
Autorité							1							1				
Reconnaissance																		
Récompense		2	1				4	1							2	2	1	1
Similarité		1	1				3							1		1		
Fiabilité							1							1				
Coopération																		
Personalisation	1			1			2		1					1				
Répétition		1					1	1								1		
Rôle social		1	1				3							1				
Expertise							1											
Sens. monde réel																		
Approb. pairs																		
Facilit. soc.																		
Verifiabilité																		
Compétition		1	1				4								1	1		

TABLE 4.2 – Nombre d'occurrences des documents associant une TP avec une ML

5. Discussions

Le nombre et la diversité des outils destinés à optimiser la QL est loin de faciliter la recherche et l'uniformité de leur utilisation. Dès lors qu'il s'agit d'y inclure une couche persuasive, nous avons vu que les principes utilisés varient selon les cas. Cette diversité des principes persuasifs disponibles et les outils dans lesquels ils sont appliqués nous montre qu'il s'agit d'un champ de recherche encore jeune pour lequel l'efficacité à long terme doit encore être validée.

Néanmoins, nous constatons que les résultats positifs qui ressortent de la littérature montrent que les TP ont un impact significatif sur les bonnes pratiques de programmation dés lors qu'elles modifient la perception et le comportement des développeurs qui y sont confrontés. Que ce soit au travers de TP ou de principes de gamification, les caractéristiques doivent cependant être analysées préalablement afin de correspondre au mieux aux outils dans lesquels elles sont déployées. Sans oublier de prendre en compte la personnalité des développeurs qui les utilisent. Notons que la majorité des groupes de test recensés dans notre corpus sont constitués d'étudiants ou de développeurs novices, souvent sur de courtes périodes. Ces résultats positifs sont donc à confirmer sur la durée.

5.1 Tendances

5.1.1 Différences entre AC et SR

Nous avons pu constater une différence significative entre les principes de conception utilisés dans les AC et dans les SR. Les AC sont principalement axés sur l'utilisation de principes influencés par la gamification comme la récompense et la compétition. Ces principes sont efficaces et correspondent bien lorsqu'il s'agit d'effectuer une action comme la correction de bugs ou la détection de CS. Les résultats de tests effectués montrent en tout cas une différence significative entre les groupes ayant effectué ce type de tâche avec un AC utilisant des TP en comparaison de ceux n'en disposant pas.

Pour les SR, conjointement aux critères de sélection retenus pour l'analyse des documents, il ressort que le concept de confiance accordé au produit joue un rôle très important. Si l'on souhaite le raccrocher à un principe persuasif existant, il pourrait être mis en comparaison avec l'étude de similarité de Stanford effectuée par Fogg [18]. Un SR sera donc plus efficace si le développeur à confiance en ce système.

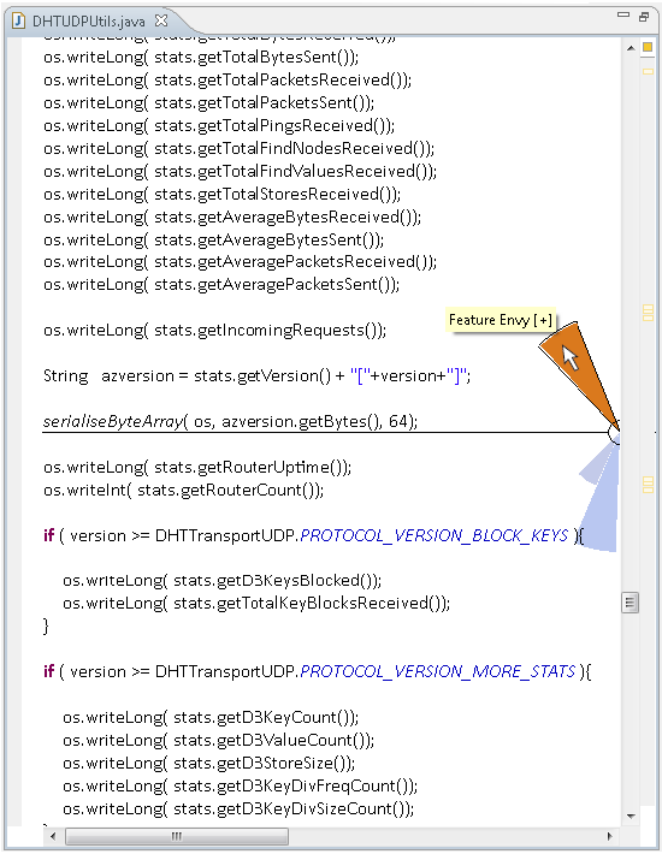

FIGURE 5.1 – Affichage interactif dans un AC [43]

5.1.2 Importance de la visualisation

En terme d'IHM, l'affichage et la mise à disposition des résultats obtenus par un AC ou un SR est un facteur persuasif d'importance qui ne doit pas être négligé. Une cartographie d'outils de visualisation peut être consultée pour s'inspirer et répertorier les éléments qui peuvent agir comme levier persuasif [40].

La visualisation et le traitement graphique des éléments peuvent faciliter le déclenchement d'un comportement cible [1]. Murphy-Hill et Black [43] proposent d'ailleurs une ligne directrice reprenant les caractéristiques les plus importantes d'un tel outil (TABLE 5.1 p.45). Ces caractéristiques renvoient pour la plupart aux principes persuasifs repris dans le PSD. Les résultats de leur expérimentation semblent confirmer que l'application systématique de ces caractéristiques, combinée à une visualisation interactive (FIGURE 5.1) calculée en tâche de fond (*i.e.* sans perturber directement l'activité principale du programmeur) a un réel potentiel.

De manière plus transversale, l'utilisation d'éléments graphiques spécifiques peut faciliter et encourager les développeurs à produire un code lisible et de qualité. Cette caractéristique esthétique, reprise comme critère d'interaction persuasive statique dans [6], peut être utilisée dans des outils de détection de CS [43] mais aussi dans les SR où la qualité de l'interface graphique peut avoir un impact important sur

1. Voir le modèle comportemental de Fogg (FIGURE 2.6 p.17)

Guideline	
Caractéristique	*Description*
Retenue	L'outil ne devrait pas submerger l'utilisateur par le nombre de CS détectés
Relation	L'outil devrait afficher le lien qui peut exister entre les différentes parties du code impactées par un CS
Partialité	L'outil devrait mettre l'accent sur les CS qui sont plus difficiles à détecter
Non distrayant	L'outil ne devrait pas distraire le développeur
Estimabilité	L'outil doit être capable d'estimer l'ampleur des CS détectés
Disponibilité	L'outil devrait mettre facilement et rapidement à disposition les informations concernant les CS détectés, sans que cela nécessite trop de manipulations
Discrétion	L'outil ne devrait pas être bloquant et devrait permettre à l'utilisateur de réaliser ses tâches normales
Sensibilité au contexte	L'outil devrait traiter en priorité les CS relatifs au contexte dans lequel se trouve l'utilisateur
Lucidité	En plus d'avertir l'utilisateur sur l'existence éventuelle de CS dans le code, l'outil devrait en expliquer l'origine et la signification

TABLE 5.1 – Ligne directrice des caractéristiques d'un système persuasif [43]

l'acceptation des recommandations [22] au même titre que le contexte de travail. Ce sont donc des caractéristiques importantes qui ressortent de notre analyse et qui sont à prendre en considération.

5.2 Risques et challenges

Globalement, les AC peuvent remonter un nombre important de problèmes ou avertissements. Cela peut poser un souci majeur car si la liste des avertissements ou erreurs potentiellement remontés par un tel outil est jugée trop grande par un développeur alors celui-ci aura tendance à hésiter à les corriger [3].

Si nous reprenons les outils spécifiquement développés dans un objectif volontaire d'intégration de TP, nous constatons qu'ils ne sont pas génériques et se limitent souvent à un IDE particulier (*i.e. Eclipse* pour le langage Java). Cette limitation est également valable pour les attributs de qualité surveillés par ces outils. Prenons un exemple avec l'outil développé cette fois dans l'environnement Visual Studio 2010 [53], qui se concentre uniquement sur la maintenabilité, laissant de coté le reste des attributs de qualité comme la fiabilité, la disponibilité et l'utilisabilité (FIGURE 5.2).

Notons également qu'il peut y avoir des critères d'interaction persuasive qui agissent en défaveur du comportement cible dans un contexte où ceux-ci seraient le plus ressentis négativement. En posant les bases de la captologie, Fogg conclut que les signaux sociaux utilisés au travers d'un ordinateur ne sont pas appropriés dans une technologie ayant pour seul objectif d'améliorer l'efficacité. Il précise que

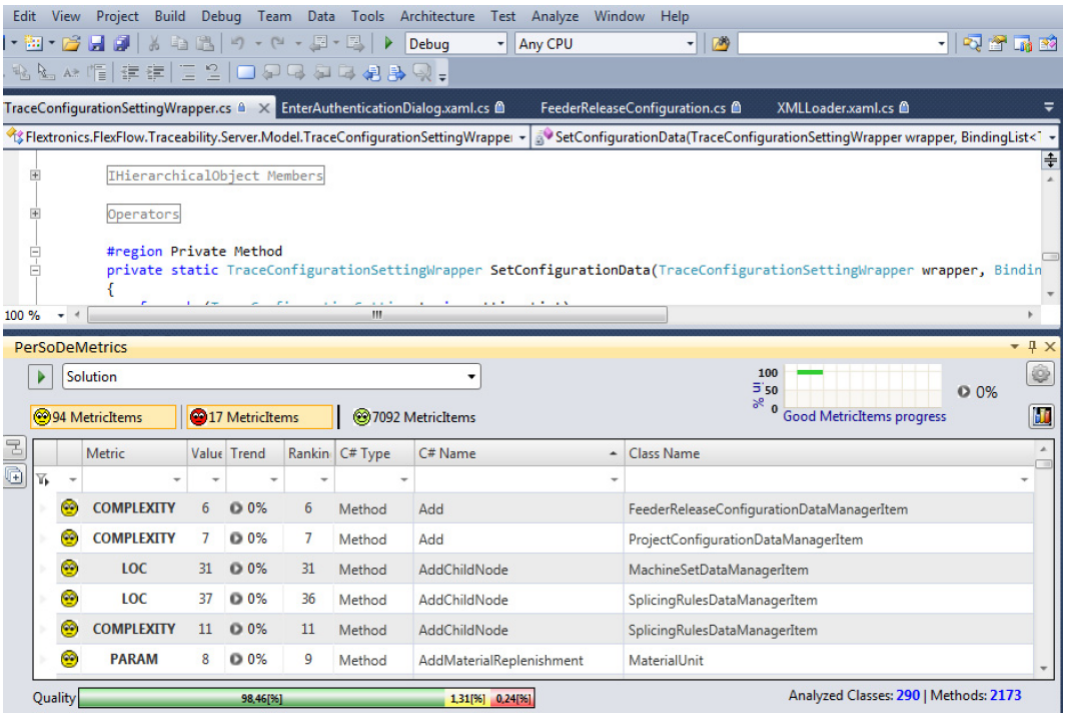

FIGURE 5.2 – Outil métrique persuasif dans Visual Studio 2010 [53]

les tâches réalisées par le biais de ces technologies seraient ralenties par ces signaux sociaux. C'est donc un point à prendre en considération lors de la conception d'un outil dont l'objectif est d'améliorer la qualité du code et par extension, l'efficacité du développeur.

5.2.1 Gamification

L'usage de récompenses constitue la pratique la plus répandue en terme de gamification de système. Les résultats obtenus par l'utilisation de récompenses sont positifs et son utilisation semble clairement être appréciée par les développeurs utilisant des outils intégrant ce principe. Sa conception et son utilisation devront cependant être surveillées afin d'éviter tout détournement par la pratique systématique d'opérations inutiles qui permettraient d'obtenir ces récompenses [41]. Le système devra donc limiter l'attribution de récompenses pour éviter ce type de comportement.

Rappelons que, conformément au modèle comportemental de Fogg, la motivation seule ne suffit pas à atteindre un comportement cible. Se pose alors la question de la véritable efficacité de la gamification à long terme si elle n'est pas utilisée conjointement aux TP énoncées initialement par Fogg [18] et développées plus spécifiquement dans le PSD [49].

Prenons l'exemple d'une tâche consistant à réduire le nombre de bugs ou de CS détectés par un AC, ce mécanisme de récompense étant le plus utilisé en réponse à l'accomplissement de cette tâche par l'utilisateur. L'utilisation de ce principe de récompense devrait donc être encouragé dans le cadre du développement d'un AC efficace. Mais comme la gamification joue principalement sur la motivation par le

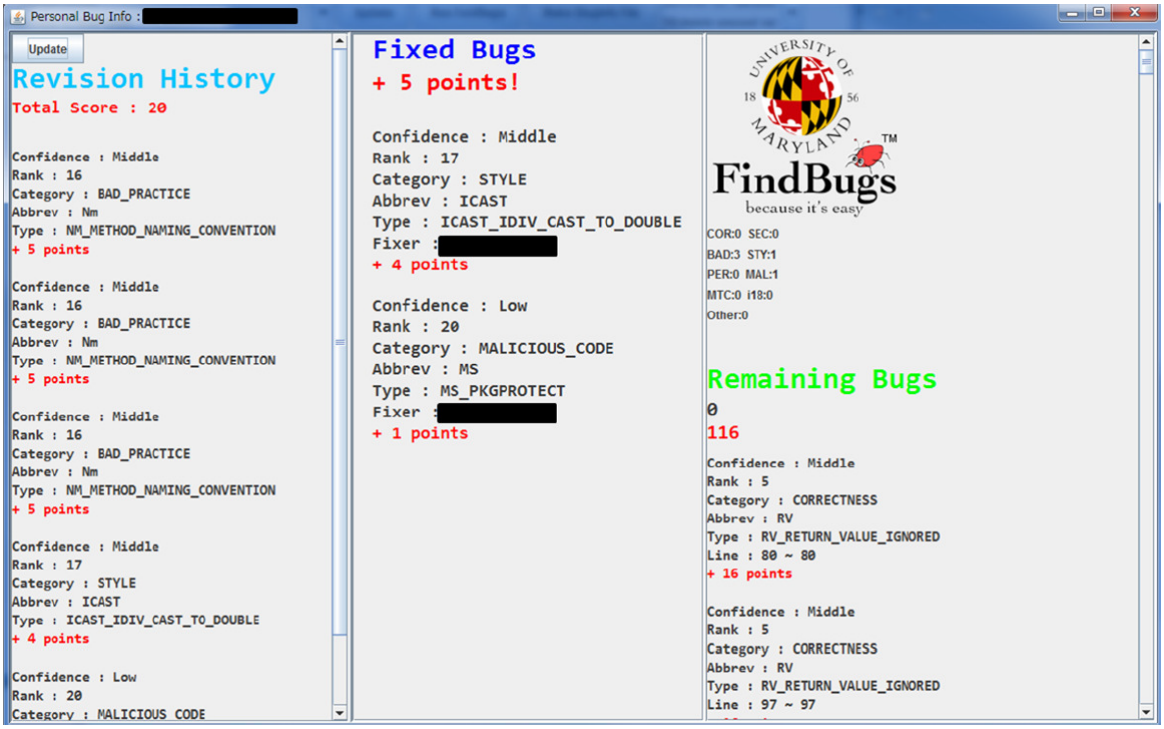

FIGURE 5.3 – Exemple de tableau de récompenses [3]

biais de ce principe, nous sommes en droit de nous poser la question suivante :

L'usage unique de la récompense suffit-il à modifier durablement le comportement d'un utilisateur pour l'amener à produire un code de meilleure qualité ?

Beaucoup d'outils se basent sur un tableau de récompenses qui donne l'opportunité de visualiser directement son niveau et sa progression dans la réalisation d'une tâche (FIGURE 5.3). Ce genre de procédé est basé sur la motivation du développeur à atteindre ses propres objectifs mais peut également déboucher sur un esprit de compétition lorsqu'il est comparé avec d'autres membres d'une équipe de développement. Dans ce cas, les éléments ludiques utilisés peuvent entraver les processus de refactorisation du code [14] voire même aboutir à des effets indésirables lorsque les principes ludiques appliqués (*e.g.* scoring, badges) dans un processus de développement provoquent une concurrence perçue négativement [52]. Il faut rester vigilant sur son utilisation si les niveaux diffèrent grandement. Cette situation risque alors d'aboutir à une baisse de la motivation à utiliser un système basé sur ce type de principe. Il faudra donc surveiller le côté clivant que ces stimuli peuvent avoir sur les utilisateurs lors de la mise en place d'un tel système.

En résumé, notons que le succès de la gamification dépend initialement de la motivation intrinsèque de l'utilisateur, sa disposition à "jouer le jeu". Comme nous venons de le décrire, s'il perd cette motivation alors tous les effets positifs de la gamification peuvent s'inverser et avoir consécutivement un effet négatif sur la volonté d'utiliser le système [1]. Il y a donc un subtil équilibre à maintenir lorsqu'on décide de *gamifier* un système.

Tous ces principes persuasifs sont applicables et peuvent s'avérer efficaces pour

modifier le comportement d'un développeur mais il faut ajouter une dimension supplémentaire pour vérifier son efficacité. Il s'agit de connaître les développeurs qui seront confrontés à ce système car selon l'expérience, la volonté d'utiliser un tel système peut être très variable.

5.3 Directions de recherche possibles

En terme d'intégration de TP dans un IDE, les futurs développements devraient s'inspirer des travaux développés dans [43], [53], [23], [22], [61].

5.3.1 Analyseur de code

Concernant les AC, nous avons relevé que les prototypes développés l'étaient majoritairement en Java, ce qui rend une généralisation de ces outils encore difficilement évaluable. Il en va de même pour l'intégration de TP dans un outil de détection de CS existant car ces derniers sont nombreux et fournissent des résultats qui peuvent grandement varier selon leur configuration (*e.g.* métriques calculées, calibrage des seuils pour ces métriques, etc.). Les résultats à long terme (*i.e.* la modification comportementale) doivent encore être validés.

5.3.2 Système de recommandation

Les SR sont à prendre en compte dans le cadre des interfaces persuasives au sein d'un IDE, notamment par leur capacité à influencer les utilisateurs sur les décisions à prendre. Cette influence dépend malgré tout de l'utilisateur car plus les développeurs gagnent en expérience, plus ils sont sujets à devenir réticents à utiliser ce type d'outil [44]. Deux facteurs d'importance ressortent dans les SR : le *contenu* et la *présentation* [61]. Ces deux facteurs sont donc également à retenir car ils constituent des candidats idéaux pour y injecter des éléments de persuasion capables de mieux agir sur le comportement à long terme du développeur (*e.g.* si le contenu proposé est validé par des experts, cela peut agir comme un rôle d'autorité).

De plus, les SR sont également très intéressants par leur aspect polyvalent. Leur contenu pourrait être alimenté par différentes sources (paramétrables selon les besoins) alors que leur interface pourrait être mutualisée pour offrir la meilleure expérience et agir de manière persuasive. Prenons un exemple, nous pourrions imaginer le développement d'un SR dont le premier objectif est d'informer l'utilisateur sur les commandes existantes d'un IDE. Ceci afin de l'aider à se familiariser avec des outils dont il n'aurait pas connaissance comme le suggèrent les articles [61], [21] et [22]. Ce même SR, et les TP qu'il embarque pour afficher ses recommandations, pourrait être réutilisé à d'autres fins dans le même IDE. Imaginons les informations remontées en tâche de fond par un AC et traitées par celui-ci. Les informations les plus pertinentes peuvent être reprises par le SR pour proposer à l'utilisateur les meilleures actions à effectuer. Ces deux types de recommandations sont différentes mais leur finalité aura pour résultat d'augmenter la QL. Nous aurions donc un SR dont le traitement et l'affichage des informations seraient génériques (en intégrant les TP les plus adéquates), peu importe la source de données traitée par ce SR.

Il ne s'agit bien évidemment là que d'idées qui émergent suite à l'analyse de notre corpus. De nouvelles expérimentations doivent encore être menées pour tenter de mutualiser l'existant sans submerger le développeur et le ralentir dans sa tâche principale, tout en validant les effets à long terme apportés par les TP sur le comportement des utilisateurs.

5.3.3 Cartographie

Dès lors, nous envisageons une recherche approfondie sur la modification comportementale résultant de l'utilisation de ces outils afin de pouvoir les catégoriser grâce à l'assistant comportemental de Fogg [16]. Nous souhaitons développer un framework qui proposerait une association entre un outil et un comportement cible pour une temporalité précise. Cela permettrait une cartographie plus exhaustive de ces outils et offrirait un moyen simple et efficace d'obtenir l'information concernant le cas précis où un outil est conseillé. Qu'il s'agisse, par exemple, de modifier un comportement une seule fois ou d'arrêter complètement un comportement existant chez un développeur, ce framework pourrait servir de guide pour choisir un outil approprié en fonction du besoin.

6. Limitations

L'analyse a été réalisée sur base d'un corpus de 35 documents. Notre recherche étant établie sur un ensemble précis de critères et principes, nous avons attentivement répertorié les références faites par les auteurs de ces documents sur base de ceux-ci. Il y a néanmoins un risque non négligeable que la terminologie utilisée ou la manière de présenter ces principes diffèrent et aient donc échappé à notre recensement. De plus, la frontière entre certains des principes se rapportant aux TP est quelquefois difficile à cerner précisément et ceux-ci peuvent même se chevaucher, ce qui peut biaiser l'analyse qui en est faite.

La littérature sur le sujet de la QL nous a fait prendre conscience qu'il n'y a pas véritablement de formalisme concernant les CS et que la qualité et la pertinence de leur détection restent encore subjectives. De même qu'il n'existe pas un moyen prouvé et efficace pour implémenter des TP, eu égard à la diversité des utilisateurs potentiels.

Le recensement des ML utilisées dans notre méthodologie a été élaboré incrémentalement en fonction du relevé effectué dans notre corpus. Il n'est donc pas basé sur une liste exhaustive de ML issues de la littérature.

7. Conclusion

Cette recherche a permis de cartographier les facteurs utilisés dans les systèmes persuasifs qui peuvent considérablement influencer le comportement des développeurs. Nous avons vu que l'usage efficace des TP dans le cadre du développement logiciel peut avoir un impact positif et renforcer la qualité d'un produit. Nous avons également vu que l'essor de la gamification et son utilisation pouvaient avoir un impact positif sur la motivation, à condition de bien gérer le dosage des principes utilisés. L'efficacité des TP dépend bien évidemment du public et du comportement cible, d'où l'intérêt d'étudier au préalable l'impact que ces technologies ont sur l'utilisateur et à travers quels outils elles sont le plus efficaces.

Nous avons mis en avant les principes de conception issus des TP les plus couramment utilisées dans le cadre de l'optimisation de la QL. Nous avons constaté qu'une grande proportion de son utilisation est influencée par la mise en place, depuis une dizaine d'années, de procédés ludiques, plus spécifiquement pour les AC. Malgré des résultats positifs dans ce domaine de recherche (*i.e.* la gamification), il existe une différence majeure avec les TP qui devraient par nature être discrètes. En effet, il ressort que la gamification s'appuie davantage sur la motivation et sur le niveau d'exigence que l'utilisateur se fixe pour atteindre ses objectifs. Il s'agira donc plus d'un choix personnel (motivation intrinsèque) dont la réalisation est stimulée par la mise en place d'éléments destinés à motiver le développeur à atteindre ses objectifs personnels (motivation extrinsèque). Les TP au sens large sont plus généralement implémentées par le concepteur du système pour amener l'utilisateur à atteindre un comportement cible, l'utilisateur n'ayant dans ce cas pas ce même besoin de motivation intrinsèque pour rendre les TP efficaces. C'est ce qui constitue d'après nous la différence majeure entre TP et gamification dans un système.

Nous avons vu que les TP implémentées dans les SR donnaient également des résultats positifs, en s'appuyant sur un système qui doit proposer une information crédible et fiable. Utilisés conjointement aux AC, leurs effets pourraient être mutualisés afin d'agir comme un déclencheur encore plus efficace pour atteindre le comportement cible. Le flux d'information très important pouvant être remonté par un AC a donc plus de chance d'être traité correctement s'il est accompagné d'un SR qui guide le développeur en lui fournissant des recommandations dont la forme et le contenu sont travaillés pour être le plus persuasifs.

Quoiqu'il en soit, la mise en pratique de telles techniques dans le cadre du développement logiciel et de l'amélioration de la qualité du code et des bonnes pratiques de programmation en général est encore peu exploitée à l'heure actuelle, offrant l'opportunité de pousser plus en avant ce champ de recherche. En effet, les tentatives

actuelles restent encore souvent à l'étape de prototype et les échantillons sur lesquels ces recherches sont effectuées ne correspondent pas toujours à la diversité existante parmi les développeurs. Les résultats observés concernant l'utilisation de TP et de gamification sont malgré tout très encourageants et ce, même si le développement logiciel représente un domaine où l'application des TP n'est pas le plus évident à mettre en oeuvre.

En poussant davantage la recherche dans ce domaine pour en valider les effets à long terme et en développant conjointement un framework permettant une cartographie précise des outils adéquats pour un comportement cible, nous pourrions sensiblement augmenter la QL grâce aux TP.

Bibliographie

[1] B. S. Akpolat et W. Slany. « Enhancing software engineering student team engagement in a high-intensity extreme programming course using gamification ». In : *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE T)*. 2014, p. 149-153.

[2] B. Amir et P. Ralph. « Proposing a Theory of Gamification Effectiveness ». In : *Companion Proceedings of the 36th International Conference on Software Engineering*. ICSE Companion 2014. Hyderabad, India : Association for Computing Machinery, 2014, p. 626-627.

[3] S. Arai et al. « A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools ». In : *2014 6th International Workshop on Empirical Software Engineering in Practice*. 2014, p. 37-42.

[4] S. Baars et S. Meester. « CodeArena : Inspecting and Improving Code Quality Metrics Using Minecraft ». In : *Proceedings of the Second International Conference on Technical Debt*. TechDebt '19. Montreal, Quebec, Canada : IEEE Press, 2019, p. 68-70.

[5] B. W. Boehm. *Software Engineering Economics*. 1st. USA : Prentice Hall PTR, 1981.

[6] E. Brangier et J. M. Christian Bastien. « 12. L'évolution de l'ergonomie des produits informatiques : accessibilité, utilisabilité, émotionnalité et influençabilité ». In : *Gérard Valléry éd., Ergonomie, conception de produits et services médiatisés*. Presses Universitaires de France, 2010.

[7] E. Brangier et A. Nemery. « Set of Guidelines for Persuasive Interfaces : Organization and Validation of the Criteria ». In : *Journal of usability studies* 9 (mai 2014), p. 105-128.

[8] E. Brangier, A. Nemery et S. Kopp. « Proposition d'une Grille de Critères d'analyse Ergonomiques Des Formes de Persuasion Interactive ». In : *Proceedings of the 22nd Conference on l'Interaction Homme-Machine*. IHM '10. Luxembourg, Luxembourg : Association for Computing Machinery, 2010, p. 153-156.

[9] P. Cremonesi, F. Garzotto et R. Turrin. « Investigating the Persuasion Potential of Recommender Systems from a Quality Perspective : An Empirical Study ». In : *ACM Trans. Interact. Intell. Syst.* 2.2 (juin 2012).

[10] S. Deterding et al. « From Game Design Elements to Gamefulness : Defining "Gamification" ». In : *Proceedings of the 15th International Academic MindTrek Conference : Envisioning Future Media Environments*. MindTrek '11. Tampere, Finland : Association for Computing Machinery, 2011, p. 9-15.

[11] S. Deterding et al. « Gamification. using game-design elements in non-gaming contexts. » In : *CHI '11 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '11. ACM, 2011, p. 2425-2428.

[12] R. G. Dromey. « A Model for Software Product Quality ». In : *IEEE Trans. Softw. Eng.* 21.2 (fév. 1995), p. 146-162.

[13] D. J. Dubois et G. Tamburrelli. « Understanding Gamification Mechanisms for Software Development ». In : *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. Saint Petersburg, Russia : Association for Computing Machinery, 2013, p. 659-662.

[14] L. Elezi et al. « A Game of Refactoring : Studying the Impact of Gamification in Software Refactoring ». In : *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP '16 Workshops. Edinburgh, Scotland, UK : Association for Computing Machinery, 2016.

[15] B.J. Fogg. « A Behavior Model for Persuasive Design ». In : *Proceedings of the 4th International Conference on Persuasive Technology*. Claremont, California, USA : ACM, 2009, 40 :1-40 :7.

[16] B.J. Fogg. *Behavior Wizard : A Method for Matching Target Behaviors with Solutions*. URL : `https://captology.stanford.edu/wp-content/uploads/2010/10/Fogg-and-Hreha-BehaviorWizard.pdf` (visité le 30/07/2020).

[17] B.J. Fogg. *Fogg Behavior Model*. URL : `https://www.behaviormodel.org/` (visité le 30/07/2020).

[18] B.J. Fogg. *Persuasive Technology : Using Computers to Change What We Think and Do*. Morgan Kaufmann Publishers Inc., 2002. Chap. 5, p. 89-120.

[19] A. Foulonneau, G. Calvary et E. Vilain. « État de l'art en conception de systèmes persuasifs ». In : t. 4. 1. Association Francophone d'Interaction Homme-Machine (AFIHM), juin 2015, p. 19-47.

[20] M. Fowler. *Refactoring : Improving the Design of Existing Code*. USA : Addison-Wesley Longman Publishing Co., Inc., 1999.

[21] M. Gasparic. « Context-Based IDE Command Recommender System ». In : *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA : ACM, 2016, p. 435-438.

[22] M. Gasparic, T. Gurbanov et F. Ricci. « Improving Integrated Development Environment Commands Knowledge with Recommender Systems ». In : *Proceedings of the 40th International Conference on Software Engineering : Software Engineering Education and Training*. ICSE-SEET '18. Gothenburg, Sweden : Association for Computing Machinery, 2018, p. 88-97.

[23] M. Gasparic et F. Ricci. « IDE Interaction Support With Command Recommender Systems ». In : *IEEE Access* 8 (2020), p. 19256-19270.

[24] M. Gasparic et al. « A Graphical User Interface for Presenting Integrated Development Environment Command Recommendations : Design, Evaluation, and Implementation ». In : *Information and Software Technology* 92 (août 2017).

[25] S. Gkika et G. Lekakos. « The persuasive role of Explanations in Recommender Systems ». In : *CEUR Workshop Proceedings* 1153 (jan. 2014), p. 59-68.

[26] *Guide to the Software Engineering Body of Knowledge*. URL : `https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3` (visité le 30/07/2020).

[27] T. Hall et al. « What Do We Know about Developer Motivation ? » In : *IEEE Software* 25.4 (2008), p. 92-94.

[28] J. Hamari, J. Koivisto et T. Pakkanen. « Do Persuasive Technologies Persuade ? - A Review of Empirical Studies ». In : t. 8462. Mai 2014, p. 118-136.

[29] J. Hamari, J. Koivisto et H. Sarsa. « Does Gamification Work ? – A Literature Review of Empirical Studies on Gamification ». In : *2014 47th Hawaii International Conference on System Sciences*. Jan. 2014, p. 3025-3034.

[30] *ISO 9126*. URL : `https://www.iso.org/fr/standards.html` (visité le 30/07/2020).

[31] A. Johnston et M. Warkentin. « The Influence of Perceived Source Credibility on End User Attitudes and Intentions to Comply with Recommended IT Actions ». In : *JOEUC* 22 (juil. 2010), p. 1-21.

[32] T.B. Jordan et al. « Designing Interventions to Persuade Software Developers to Adopt Security Tools ». In : *Proceedings of the 2014 ACM Workshop on*

Security Information Workers. SIW '14. Scottsdale, Arizona, USA : ACM, 2014, p. 35-38.

[33] S. Khandelwal, S. K. Sripada et Y. R. Reddy. « Impact of Gamification on Code Review Process : An Experimental Study ». In : *Proceedings of the 10th Innovations in Software Engineering Conference*. ISEC '17. Jaipur, India : Association for Computing Machinery, 2017, p. 122-126.

[34] B. A. Kitchenham, D. Budgen et P. Brereton. *Evidence-based software engineering and systematic reviews*. T. 4. CRC press, 2015.

[35] B. Kitchenham et S. Charters. *Guidelines for performing systematic literature reviews in software engineering*. Rapp. tech. Software Engineering Group, School of Computer Science et Mathematics, Keele University, Keele, 9 July, 2007.

[36] M. Lanza et R. Marinescu. *Object-Oriented Metrics in Practice : Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Jan. 2006.

[37] M. Lanza et al. « CodeCrawler : An Information Visualization Tool for Program Comprehension ». In : *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05. St. Louis, MO, USA : Association for Computing Machinery, 2005, p. 672-673.

[38] W. Li et al. « Design and Evaluation of a Command Recommendation System for Software Applications ». In : *ACM Trans. Comput.-Hum. Interact.* 18 (juin 2011), p. 6.

[39] J. McCall. *Factors in Software Quality : Preliminary Handbook on Software Quality for an Acquisiton Manager*. T. 1-3. ADA049055. General Electric, nov. 1977.

[40] L. Merino et al. « VISON : An Ontology-Based Approach for Software Visualization Tool Discoverability ». In : *2019 Working Conference on Software Visualization (VISSOFT)*. Sept. 2019, p. 45-55.

[41] Q. Mi et al. « A Gamification Technique for Motivating Students to Learn Code Readability in Software Engineering ». In : *2018 International Symposium on Educational Technology (ISET)*. 2018, p. 250-254.

[42] E. Mourão et al. « On the performance of hybrid search strategies for systematic literature reviews in software engineering ». In : *Information and Software Technology* (2020), p. 106294.

[43] E. Murphy-Hill et A. P. Black. « An Interactive Ambient Visualization for Code Smells ». In : *Proceedings of the 5th International Symposium on Software Visualization*. SOFTVIS '10. Salt Lake City, Utah, USA : Association for Computing Machinery, 2010, p. 5-14.

[44] E. Murphy-Hill, R. Jiresal et G.C. Murphy. « Improving Software Developers' Fluency by Recommending Development Environment Commands ». In : *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. FSE '12. Cary, North Carolina : ACM, 2012, 42 :1-42 :11.

[45] G.C. Murphy et E. Murphy-Hill. « What is Trust in a Recommender for Software Development ? » In : *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. RSSE '10. Cape Town, South Africa : Association for Computing Machinery, 2010, p. 57-58.

[46] T. Nyström. « Gamification of persuasive systems for sustainability ». In : *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*. Déc. 2017, p. 1-3.

[47] J. O'Donovan et B. Smyth. « Trust in Recommender Systems ». In : *Proceedings of the 10th International Conference on Intelligent User Interfaces*. IUI '05. San Diego, California, USA : Association for Computing Machinery, 2005, p. 167-174.

[48] H. Oinas-Kukkonen. « A foundation for the study of behavior change support systems ». In : *Personal and Ubiquitous Computing* 17 (août 2013).

[49] H. Oinas-Kukkonen et M Harjumaa. « Towards Deeper Understanding of Persuasion in Software and Information Systems ». In : *First International Conference on Advances in Computer-Human Interaction*. Mar. 2008, p. 200-205.

[50] H. Oinas-Kukkonen et M. Harjumaa. « Persuasive Systems Design : Key Issues, Process Model, and System Features ». In : *Communications of the Association for Information Systems* 24 (mar. 2009).

[51] L. Ponzanelli. « Holistic Recommender Systems for Software Engineering ». In : *Companion Proceedings of the 36th International Conference on Software Engineering*. ICSE Companion 2014. Hyderabad, India : Association for Computing Machinery, 2014, p. 686-689.

[52] C. R. Prause et M. Jarke. « Gamification for Enforcing Coding Conventions ». In : *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy : Association for Computing Machinery, 2015, p. 649-660.

[53] I. Pribik et A. Felfernig. « Towards Persuasive Technology for Software Development Environments : An Empirical Study ». In : *Proceedings of the 7th International Conference on Persuasive Technology : Design for Health and Safety*. PERSUASIVE'12. Berlin, Heidelberg : Springer-Verlag, 2012, p. 227-238.

[54] F. Ricci, L. Rokach et B. Shapira. *Recommender Systems Handbook*. 1st. Berlin, Heidelberg : Springer-Verlag, 2010.

[55] R. Robbes. « On the Evaluation of Recommender Systems with Recorded Interactions ». In : *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*. SUITE '09. USA : IEEE Computer Society, 2009, p. 45-48.

[56] H. M. dos Santos et al. « CleanGame : Gamifying the Identification of Code Smells ». In : *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. SBES 2019. Salvador, Brazil : Association for Computing Machinery, 2019, p. 437-446.

[57] D. Silva, R. Terra et M. T. Valente. « Recommending Automated Extract Method Refactorings ». In : *Proceedings of the 22nd International Conference on Program Comprehension*. ICPC 2014. Hyderabad, India : Association for Computing Machinery, 2014, p. 146-156.

[58] I. Sommerville. *Software Engineering*. 9th. USA : Addison-Wesley Publishing Company, 2010.

[59] K. Torning et H. Oinas-Kukkonen. « Persuasive System Design : State of the Art and Future Directions ». In : *Proceedings of the 4th International Conference on Persuasive Technology*. Persuasive '09. Claremont, California, USA : Association for Computing Machinery, 2009.

[60] B. Vanderose. « Supporting a model-driven and iterative quality assessment methodology : The MoCQA framework ». Thèse de doct. Déc. 2012.

[61] P. Viriyakattiyaporn et G.C. Murphy. « Challenges in the user interface design of an IDE tool recommender ». In : *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. Mai 2009, p. 104-107.

[62] J. Webster et R. T. Watson. « Analyzing the Past to Prepare for the Future : Writing a Literature Review ». In : *MIS Q.* 26 (2002).

[63] R. Wettel et M. Lanza. « CodeCity : 3D Visualization of Large-Scale Software ». In : *Companion of the 30th International Conference on Software Engineering*. ICSE Companion '08. Leipzig, Germany : Association for Computing Machinery, 2008, p. 921-922.

[64] J. Wu et al. « How Is Code Recommendation Applied in Android Development : A Qualitative Review ». In : *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. 2016, p. 30-35.