ÉTUDE DE L'IMPACT DE DIFFÉRENTES SOURCES D'INFORMATIONS SUR LA SYNTHÈSE DES ENTITÉS DE CODE LOGICIEL

par

Hafida HAKIMI

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION
DE LA MAÎTRISE AVEC MÉMOIRE EN GÉNIE LOGICIEL
M.Sc.A.

MONTRÉAL, LE "21 DÉCEMBRE 2017"

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Mme. Latifa GUERROUJ, Directrice de Mémoire
Département de génie logiciel et des technologies de l'information, École de technologie supérieure (ÉTS).

M. Luc DUONG, Président du Jury
Département de génie logiciel et des technologies de l'information, École de technologie supérieure (ÉTS).

M. Ali OUNI, membre du jury
Département de génie logiciel et des technologies de l'information, École de technologie supérieure (ÉTS).

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE "19 DÉCEMBRE 2017"

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Tout d'abord, mes remerciements vont tout particulièrement à ma Directrice de mémoire, Madame Latifa Guerrouj, pour l'intérêt qu'elle a porté à cette étude, les critiques et les conseils hautement bénéfiques, la confiance, la disponibilité et le soutien moral et financier qu'elle m'a accordés tout au long de ce travail de recherche. Qu'elle trouve ainsi, l'expression sincère et concrète de ma reconnaissance et de ma haute gratitude.

Je remercie les membres du jury, Professeur Luc Duong et Professeur Ali OUNI, qui me feront l'honneur d'évaluer ce modeste travail.

Je remercie les professeures Olga Baysal (Université de Carleton, Ottawa, Canada) et Bonita Sharif (Université de Youngstown State, Ohio, États-Unis) pour leurs contributions et aides précieuses à la réalisation de cette étude.

J'exprime vivement ma gratitude à l'ensemble des enseignants qui m'ont suivi inlassablement durant tout mon cursus universitaire.

Je ne peux pas oublier de remercier toute personne qui a contribué de prés ou de loin à la réussite de ce travail, particulièrement les participants de l'expérimentation contrôlée.

Finalement, je tiens à remercier mes chers parents, mon mari et mes sœurs pour leurs soutiens tout au long de mes études universitaires.

Je dédie ce mémoire à ma famille, principalement à mes exceptionnels parents, à mon fabuleux mari, à mes adorables soeurs particulièrement Rania, à ma très chère grand-mère Mamanto ainsi qu'à tous mes amis.

ÉTUDE DE L'IMPACT DE DIFFÉRENTES SOURCES D'INFORMATIONS SUR LA SYNTHÈSE DES ENTITÉS DE CODE LOGICIEL

Hafida HAKIMI

RÉSUMÉ

Les développeurs logiciel mentionnent ou discutent des entités/éléments de code (méthodes et classes) dans plusieurs sources d'informations logiciel telles que le code source, le site Stack Overflow, les discussions de développeurs, les rapports de bogues, etc. Pour aider les développeurs à comprendre rapidement les éléments de code, des recherches antérieures ont suggéré la synthèse de code. Alors que plusieurs travaux sur la synthèse de code ont utilisé le code source pour synthétiser les classes et les méthodes, il existe très peu de travaux menés sur l'utilisation de la documentation informelle pour la synthèse des éléments de code.

Dans ce présent mémoire, nous étudions l'utilisation de quatre différents types de sources d'informations pour synthétiser la mise en œuvre et l'utilité des classes et des méthodes. Les sources d'informations examinées comprennent le code source, le Stack Overflow, les rapports de bogues et la combinaison des trois sources d'informations (code source, Stack Overflow et les rapports de bogues). Nous avons mené une étude auprès de 24 développeurs professionnels issus de différentes entreprises et institutions canadiennes. Les développeurs ont été invités à effectuer des tâches de synthèse de quatre éléments de code différents en utilisant le code source, le Stack Overflow, les rapports de bogues ou la combinaison de ces trois sources d'informations. Ces quatre éléments de code sont choisis parmi un ensemble de huit éléments de code issus de quatre projets *Java* open-source : Eclipse, JMeter, Tomcat, Netbeans.

Les synthèses fournies par les développeurs professionnels ont été évaluées par rapport à des synthèses fournies par des annotateurs.

Les résultats montrent que la performance des développeurs professionnels lors de la synthèse des éléments de code en termes de précision, rappel et F-mesure est meilleure par l'utilisation du code source et du Stack Overflow. Cependant, il n'existe pas de différences statistiquement significatives en termes de performances des développeurs professionnels lors de l'utilisation des quatre différents types de sources d'informations logiciel. Nous pouvons donc conclure que la documentation informelle (c'est-à-dire le Stack Overflow, les rapports de bogues ou la combinaison des sources d'informations logiciel) est aussi importante que le code source et qu'elle peut être utilisée pour effectuer la tâche de synthèse d'éléments de code.

Ces résultats sont intéressants et peuvent être utiles pour déterminer les informations pertinentes à inclure lors de la création d'outils de synthèse d'éléments de code.

Comme futurs travaux, il serait intéressant de mettre l'accent sur la stratégie adoptée par les développeurs lors de la synthèse des éléments de code lorsqu'ils utilisent du code source et la documentation informelle, et de définir les parties (c'est-à-dire, le texte en langage naturel, les

exemples de code, les traces de pile, etc.) dont les développeurs tirent le plus d'informations lors de la préformation des tâches de synthèse de code.

Mots-clés: Éléments de code, Synthèse, Code source, Stack Overflow, rapport de bogues, Documentation informelle, Documentation formelle, Expérimentation contrôlée, Développeurs professionnels.

STUDY OF THE IMPACT OF DIFFERENT SOURCES OF INFORMATION ON THE SYNTHESIS OF SOFTWARE CODE ELEMENTS

Hafida HAKIMI

ABSTRACT

Software developers mention or discuss code elements (methods and classes) from multiple sources of software information such as source code, stack overflow, developer discussions, bug reports, etc. In order to help developers quickly understand code elements, previous research works have suggested code synthesis. Several works on code synthesis has used source code to synthesize classes and methods, there is a few work on using informal documentation to summarize code elements.

In this thesis, a study is conducted on the use of four different types of information sources to synthesize the implementation and usefulness of classes and methods. Sources of information examined includes source code, Stack Overflow, bug reports, and the combination of these three sources. A study was conducted with 24 professional developers from different Canadian companies and institutions. The developers were asked to perform tasks that summarize four different parts of the code using source code, stack overflow, bug reports, or the combination of these three sources of information. These four pieces of code are chosen from a set of code elements from four open-source projects *Java* : Eclipse, JMeter, Tomcat, Netbeans.

The summaries provided by the professional developers were evaluated against the summaries provided by three annotators.

The results show that developers are better synthesized when they use source code and Stack Overflow. However, there are no statistically significant differences in the accuracy of syntheses provided by developers when using four different sources of information. An empirical evidence states that informal documentation (such as Stack Overflow or bug reports) is as important as the source code and can be used to perform the code summarization task.

These results are interesting and can be useful in determining the relevant information when creating powerful and efficient automatic code-element summarization tools.

As a future work, it would be interesting to focus on the strategy adopted by developers when summarizing code elements using source code and informal documentation, and defining the parts (natural language text, code samples, stack traces, etc.) where developers get the most information from reproducing code synthesis tasks.

Keywords: Code elements, Summary, Source code, Stack overflow, Bug report, Informal documentation, Formal documentation, Controlled experimentation, Professional developers.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

API	Application Programming Interface / Interface de programmation applicative
CS	Code source
EDI	Environnement de développement intégré
ETS	École de Technologie Supérieure
$H_{0,QRX}$	Hypothèse nulle relative à la question de recherche X
$H_{a,QRX}$	Hypothèse alternative relative à la question de recherche X
H_{0,QRX_Y}	Hypothèse nulle relative à la question de recherche spécifique Y liée à la question de recherche X
H_{a,QRX_Y}	Hypothèse alternative relative à la question de recherche spécifique Y liée à la question de recherche X
QRX	Question de Recherche numéro X
ID	Identifiant
QRX_Y	Question de recherche spécifique Y de la question de recherche X
RB	Rapports de Bogues
SO	Stack Overflow

INTRODUCTION

Contexte et problématique

Les développeurs doivent comprendre les éléments de code qui font partie de leurs tâches lorsqu'ils exécutent des activités de maintenance et d'évolution logicielle. Pour comprendre les éléments de code qui les rendent perplexes, les développeurs se tournent vers différents types de sources d'informations, notamment le code source, la documentation officielle ou la documentation informelle comme Stack Overflow, les rapports de bogues, les discussions des développeurs, etc.

Lire une grande quantité de code ou de texte peut être difficile et peut prendre beaucoup de temps. Pour aider les développeurs dans cette tâche non triviale, la synthèse de code a été suggérée comme un moyen pour fournir aux développeurs des informations pertinentes sur le code.

Bien qu'il y ait eu des travaux de recherche intéressants sur la synthèse de code, au meilleur de nos connaissances, les recherches antérieures se sont principalement concentrées sur le code source pour synthétiser les méthodes et les classes (Haiduc *et al.*, 2010; Moreno *et al.*, 2013a; Sridhara *et al.*, 2010; McBurney & McMillan, 2014; Moreno *et al.*, 2015). Par exemple, les auteurs Moreno *et al.* (2013a) ont généré des synthèses de classes *Java* en exploitant le contenu et les responsabilités des classes pendant que les chercheurs Sridhara *et al.* (2010) se sont appuyés sur des informations pertinentes provenant des instructions des méthodes pour fournir des synthèses en langage naturel. D'autres chercheurs tels que McBurney & McMillan (2014) ont examiné le contexte en analysant comment les méthodes *Java* sont invoquées pour générer des synthèses pour les méthodes *Java*.

Plus récemment, les chercheurs Guerrouj *et al.* (2015) ont étudié le contenu du site Stack Overflow pour la synthèse des éléments de code contenus dans les publications Stack Overflow. Ils ont considéré comme contexte l'information qui entoure les classes ou les méthodes.

La nouveauté de notre travail de recherche est l'étude des différents types d'informations et de leur pertinence pour la synthèse d'éléments de code. À cet effet, nous avons mené une expérimentation contrôlée auprès de 24 développeurs logiciel, provenant de 16 sociétés et institutions canadiennes : CGI, Deloitte, la société Ciena, B-CITI, Fujitsu Conseil Canada, Gestisoft, 360Medlink, Alithya, Banque Nationale du Canada, Rideau Recognition Solutions, GIRO, Ooda Technologies, SecureOps, CAE, PSP Investments et ministère des Services partagés Canada, branche : Superinformatique. Les développeurs ont été invités à effectuer des tâches de synthèse de quatre éléments de code différents en utilisant le code source, le Stack Overflow, les rapports de bogues ou la combinaison des trois sources d'informations (code source, le Stack Overflow, les rapports de bogues). Les quatre éléments de code appartiennent à un ensemble de huit éléments de code choisis aléatoirement parmi quatre projets *Java* open-source : Eclipse, JMeter, Tomcat et NetBeans.

Les synthèses générées par les développeurs professionnels ont été évaluées par rapport aux synthèses réalisées par les trois annotateurs, dont l'auteure de ce présent mémoire ainsi que deux professeures Olga Baysal (Université de Carleton, Ottawa, Canada) et Bonita Sharif (Université de Youngstown State, Ohio, États-Unis). De plus, nous avons comparé les performances des développeurs professionnels en termes de précision, rappel et F-mesure de synthèse et de temps passé à effectuer les tâches de synthèse en exploitant les différents types de sources d'informations considérées.

La documentation officielle manque toujours de contexte quand il s'agit de l'utilisation de la classe ou de la méthode (Treude & Robillard, 2016). Nos résultats peuvent être exploités par des chercheurs et des praticiens qui sont intéressés par la construction d'outils de synthèse automatiques de code en langage naturel, et qui savent quel type de sources d'informations sont pertinentes lors de l'exécution de la tâche de synthèse.

Les outils de synthèse de code automatiques et contextuels peuvent aider à enrichir la documentation officielle en fournissant des synthèses qui décrivent non seulement la partie implémentation d'un élément de code mais aussi son utilisation, en plus d'informations pertinentes

et context-aware comme ses dépendances, etc. De tels outils de synthèse peuvent être intégrés dans les environnements de développement intégré comme *Eclipse* pour aider les développeurs à comprendre rapidement les éléments de code qui font partie de leur espace de travail.

Objectif

L'objectif de cette expérimentation contrôlée consiste (i) à déterminer si la synthèse de code peut être effectuée à partir de la documentation informelle, telle que le Stack Overflow et les rapports de bogues plutôt que de n'utiliser que des sources d'informations logiciel formelles telles que le code source et (ii) quel type de sources d'informations est le plus pertinent pour la synthèse de code.

Méthodologie de recherche

Notre méthodologie de recherche est une expérimentation contrôlée qui implique 24 développeurs professionnels issus de 16 entreprises et institutions canadiennes, dont CGI, Deloitte, la société Ciena, B-CITI, Fujitsu Conseil Canada, Gestisoft, 360Medlink, Alithya, Banque Nationale du Canada, Rideau Recognition Solutions, GIRO, Ooda Technologies, SecureOps, CAE, PSP Investments et ministère des Services partagés Canada, branche : Superinformatique. Nous avons recueilli des informations sur tous les participants de l'expérimentation contrôlée, en utilisant un pré-questionnaire dans lequel nous avons demandé aux participants d'auto-évaluer leur niveau de programmation *Java*, expérience industrielle, degré d'études, familiarité avec les tâches de maintenance et d'évolution, etc. Puis, nous avons demandé aux participants de synthétiser quatre éléments de code extraits d'un ensemble de huit éléments de code. Ces huit éléments de code sont choisis aléatoirement de quatre projets open-source *Java* : Eclipse, JMeter, Tomcat, Netbeans, qui sont considérés comme étant l'objet de cette étude. Les quatre éléments de code se composent de deux classes et deux méthodes. Les participants synthétisent les éléments de code en utilisant différentes sources d'informations (code source, Stack Overflow, rapports de bogues et la combinaison des trois sources d'informations). Les synthèses sont recueillies par le biais de questionnaires expérimentaux. Par la suite, nous avons présenté

aux participants un post-questionnaire, qui récolte tous les commentaires pertinents liés à l'expérimentation contrôlée pouvant remplir les deux questionnaires précédents.

Enfin, nous avons mesuré la performance des développeurs professionnels en termes de précision, de rappel et de F-mesure de leurs synthèses par rapport à des synthèses de trois annotations humains, dont l'auteure de ce présent mémoire ainsi que deux professeures Olga Baysal (Université de Carleton, Ottawa, Canada) et Bonita Sharif (Université de Youngstown State, Ohio, États-Unis).

Organisation du mémoire

Le chapitre 1 présente une revue de la littérature sur les travaux liés à notre étude. Le chapitre 2 décrit l'expérimentation contrôlée effectuée, les questions de recherche, les hypothèses et les variables de recherche, ainsi que la planification de l'expérimentation contrôlée et la description des méthodes d'analyse appliquées. Le chapitre 3 comporte la présentation et la discussion des résultats de l'expérimentation, les obstacles à la validité et les limites de l'approche. La conclusion et les perspectives du projet sont présentées dans le dernier chapitre. Nous avons mis en annexes des informations supplémentaires sur l'expérimentation contrôlée menée dans notre étude.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

Ce chapitre présente les principales contributions liées à la synthèse des éléments de code logiciel dans la littérature. Ce chapitre est réparti en deux parties, dont la première partie consiste à présenter les travaux liés directement à la synthèse des éléments de code logiciel. Puis la deuxième partie présente les travaux annexes à la synthèse des éléments de code, dont ceux menés sur les APIs (Application Programming Interface). Nous mettons l'emphase sur les APIs, car nos éléments de code consistent en des classes et méthodes d'APIs.

1.1 Synthèse des éléments de code logiciel

Dans cette section, les principaux travaux liés à la synthèse des éléments de code logiciel sont présentés chronologiquement, allant des premières contributions jusqu'aux plus récentes. Parmi les pionniers en synthèse de code source, Professeure Murphy (1996) a proposé deux techniques pour synthétiser le code source. La première technique permet de synthétiser les informations structurelles dans un contexte d'un modèle de haut niveau. La deuxième technique soutient le processus de synthèse en facilitant la numérisation et l'analyse des sources d'informations d'un système, spécifiquement pour l'information structurelle difficile à extraire.

Jusqu'en 2007, la synthèse automatique du texte en langage naturel a été largement étudiée et de nombreuses approches ont été proposées, elles reposent principalement sur la récupération de texte, l'apprentissage machine et les techniques de traitement du langage naturel (Jones, 2007). Puis, les auteurs Carenini *et al.* (2007) ont proposé un cadre pour la synthèse des courriels échangés entre développeurs. Par l'utilisation d'un graphe de fragments pour essayer de détecter une conversation par courriel et en considérant des mots-clés spécifiques, afin de mesurer l'importance des phrases dans la synthèse des courriels. Ils présentent une comparaison de cette approche avec diverses méthodes existantes.

Une reproduction du concept de synthèse automatique s'est vue appliquer à la documentation formelle dans les recherches menées par les auteurs Haiduc *et al.* (2010), afin de faciliter sa compréhension aux développeurs, où ils ont proposé une solution qui obtient des descriptions textuelles. Ces descriptions illustrent les éléments de code logiciel sans avoir à lire les détails. Les auteurs ont considéré le code source comme étant une des sources d'informations utilisées pour la synthèse.

D'autre part, la synthèse automatique de la documentation informelle a fait principalement son apparition à travers l'étude des chercheurs Rastkar *et al.* (2010), où les auteurs ont considéré la synthèse par l'analyse des rapports de bogues. Ils ont comparé les générateurs basés sur les conversations avec les générateurs aléatoires et ont prouvé qu'un générateur formé spécifiquement sur les rapports de bogues peut être statistiquement meilleur que les générateurs de conversation existants. Les résultats prometteurs de cette étude sont une des motivations qui nous a mené à considérer les rapports de bogues dans notre présente étude.

Autres travaux menés par la chercheuse Rastkar (2010) qui est l'étude d'un cadre pour synthétiser les problèmes liés aux logiciels afin d'augmenter le niveau d'abstraction et d'améliorer la productivité des développeurs logiciel.

Les commentaires du code source ont aussi été traités comme documentation informelle. Les auteurs Sridhara *et al.* (2010) ont considéré l'information textuelle contenue dans les sources d'informations logiciel, en synthétisant par le traitement des commentaires du code source. Entre autres, les auteurs ont présenté une nouvelle technique pour générer automatiquement des commentaires récapitulatifs pour les méthodes *Java*.

Par la suite, les auteurs Moreno & Aponte (2012) ont prouvé l'importance de la synthèse des sources d'informations logiciel dans le domaine de l'ingénierie logicielle. Pour cela, ils ont mené une étude empirique où ils ont analysé la synthèse de code source. La synthèse a été produite manuellement par un groupe de développeurs *Java*, afin d'évaluer certaines techniques de synthèse basées sur la récupération de texte. Les auteurs ont décrit les caractéristiques prin-

cipales de la synthèse, quel type d'informations devrait être inclus et la qualité atteinte par certaines méthodes automatiques.

Les auteurs Moreno *et al.* (2013a) ont présenté une approche pour générer automatiquement une synthèse compréhensible pour les classes *Java*, en supposant qu'aucune documentation n'existe. Les synthèses permettent aux développeurs de comprendre l'objectif principal et la structure de la classe. Cette synthèse est axée sur le contenu et les responsabilités des classes, plutôt que sur leurs relations avec d'autres classes. L'outil de synthèse détermine les stéréotypes des classes et des méthodes et les utilise conjointement avec des heuristiques, pour sélectionner les informations à inclure dans la synthèse.

En ce qui concerne l'importance du contexte lors de la synthèse des éléments de code, nous retrouvons dans la thèse du professeure Guerrouj (2013) l'importance des informations contextuelles pour la normalisation du vocabulaire du code source. En comparant les résultats obtenus de deux expérimentations, qui traitent l'implication du contexte lors de la normalisation du vocabulaire. D'où l'idée d'utiliser le contexte au cours de notre expérimentation.

Au cours de la même année, les auteurs Rastkar & Murphy (2013) ont proposé l'utilisation de techniques de synthèse multi-documents pour générer une description concise en langage naturel qui explique pourquoi le code a changé afin qu'un développeur puisse choisir le bon plan d'action.

Entre autres, les auteurs Moreno *et al.* (2013b) ont proposé un plug-in Eclipse pour générer automatiquement des synthèses de langages naturels de classes *Java*. La synthèse est basée sur le stéréotype de la classe. L'outil utilise un ensemble d'heuristiques prédéfinies pour déterminer quelles informations seront utiles à la synthèse et il utilise des techniques de traitement et de génération de langage naturel pour former la synthèse. Les synthèses générées peuvent être utilisées pour ré-documenter le code et pour aider les développeurs à comprendre plus facilement les classes complexes.

Plusieurs auteurs ont étudié la synthèse des exemples de codes, dont les auteurs (i) Ying & Robillard (2013) qui ont mené une étude sur la faisabilité de synthétiser des exemples de codes, par un algorithme basé sur l'apprentissage machine qui pourrait approximer les synthèses d'un oracle généré manuellement par les humains. (ii) Les auteurs Ying & Robillard (2014) ont établi dans une liste de pratiques suivies pour synthétiser des exemples de codes et proposer des hypothèses soutenues par des principes empiriques justifiants l'utilisation de ces pratiques.

Les auteurs Moreno *et al.* (2014) ont proposé une approche pour la génération automatique des modifications et des synthèses de code source, provenant de systèmes de version et de suivi des problèmes. L'approche a été conçue en fonction de l'analyse manuelle de 1 000 notes de versions existantes. Pour l'évaluation de la qualité des notes de version de l'approche, ils ont effectué trois études empiriques impliquant 53 participants (45 développeurs professionnels et 8 étudiants).

Récemment, une autre source d'informations a été considérée par les auteurs Guerrouj *et al.* (2015) qui est le Stack Overflow. Les auteurs ont généré automatiquement des synthèses d'éléments de code discutés dans le Stack Overflow. Ces synthèses ont été évaluées par rapport à des synthèses fournies par deux annotateurs. Les résultats de l'étude montrent que l'approche atteint un certain taux de précision, ce qui prouve l'importance de cette source d'informations dans le cadre de notre étude.

De plus, les chercheurs Ponzanelli *et al.* (2015) ont présenté une approche pour compléter les techniques de synthèse existantes pour faire face à la nature hétérogène et multidimensionnelle des sources d'informations complexes.

Enfin, les auteurs McBurney & McMillan (2016) ont suggéré une approche pour la génération automatique des méthodes *Java*. Ils ont synthétisé le contexte entourant la méthode en utilisant le code source comme source d'informations. Ils ont montré une certaine amélioration de la qualité de synthèse. Dans notre approche, nous étudions les classes et les méthodes comme éléments de code.

1.2 Travaux menés sur les APIs

Dans cette section, les principaux travaux liés à la synthèse, la documentation et les informations menées sur les APIs sont présentés chronologiquement, allant des premières contributions jusqu'aux plus récentes. Nous mettons l'emphase sur les APIs, car nos éléments de code consistent en des classes et méthodes d'APIs.

Les chercheurs Zibran *et al.* (2011) ont identifié les problèmes d'utilisabilité des APIs qui se reflètent dans les messages de bogues des utilisateurs et distinguent la signification relative des facteurs d'utilisabilité. De plus, ils ont pu identifier les problèmes d'utilisabilité de l'API les plus fréquents.

Par la suite, les auteurs Duala-Ekoko & Robillard (2012) ont identifié vingt différents types de questions que les programmeurs demandent lorsqu'ils travaillent avec des APIs et fournissent de nouvelles idées sur la cause des difficultés rencontrées par les programmeurs lorsqu'ils répondent à des questions sur l'utilisation des APIs afin d'améliorer l'apprentissage et à identifier les domaines du processus d'apprentissage des APIs.

En parallèle, les auteurs Dagenais & Robillard (2012) ont proposé une technique qui identifie les termes de type code dans les documents et relient ces termes à des éléments de code spécifiques des APIs.

Plusieurs travaux ont été menés sur la documentation des APIs. Par exemple, les auteurs Subramanian *et al.* (2014) qui ont décrit une méthode itérative et déductive de liaison de code source à la documentation de l'API. Les auteurs ont proposé une implémentation de cette méthode, qui prend en charge les langages de programmation *Java* et *JavaScript*. Ils ont prouvé que cette implémentation améliore la documentation des APIs traditionnelles avec du code source mis à jour ; elle peut également être utilisée pour incorporer des liens vers la documentation des APIs dans les extraits de code source. Les auteurs Uddin & Robillard (2015) ont étudié comment 10 problèmes communs de documentation se sont manifestés dans la pratique. Les résultats sont basés sur deux sondages sur un total de 323 développeurs logiciel professionnels et une

analyse de 179 unités de documentation API. Les trois problèmes majeurs étaient l'ambiguïté, l'incomplétude et l'inexactitude du contenu. Les répondants ont souvent mentionné six des 10 problèmes qui les obligeaient à utiliser une autre API. Les auteurs Robillard & Chhetri (2015) ont proposé de détecter et de recommander des fragments de documentation API potentiellement importants pour un programmeur qui a déjà décidé d'utiliser un certain élément API. Ils classifient les fragments de texte dans la documentation de l'API en fonction de l'information indispensable. Ils extraient des modèles de mots et utilisent ces modèles pour trouver automatiquement de nouveaux fragments.

Plus récemment, les auteurs Treude & Robillard (2016) ont combiné la documentation des APIs avec les informations contenues dans le Stack Overflow. Les auteurs présentent une approche pour améliorer automatiquement la documentation des APIs à partir de phrases tirées du Stack Overflow et liées à un type de code particulier et qui fournissent un aperçu non contenu dans la documentation de l'API. Les auteurs ont pu montrer que les méta-données disponibles sur le Stack Overflow peuvent améliorer considérablement les approches d'extraction d'informations lorsqu'elles sont appliquées aux données contenues dans le Stack Overflow.

D'autres chercheurs ont considéré le Stack Overflow dans l'extraction d'informations sur les APIs, tels que les auteurs Venkatesh *et al.* (2016) qui ont su extraire des informations sur les APIs à partir de forums de développeurs et du Stack Overflow. En prenant en considération un certain nombre de codes discutés dans le Stack Overflow ainsi que dans les discussions de forums dédiés aux développeurs.

CHAPITRE 2

DÉFINITION ET PLANIFICATION DE L'EXPÉRIMENTATION CONTRÔLÉE

Ce chapitre est composé de trois parties distinctes, (i) la définition de l'expérimentation contrôlée menée dans cette étude, (ii) sa planification et (iii) la méthode d'analyse des données récoltées. Nous présentons en première partie l'expérimentation contrôlée menée dans notre étude en définissant : l'objectif, le "quality focus", la perspective, le contexte, les sujets et les objets étudiés. Ensuite, nous formulons les questions et hypothèses de recherche. Puis, nous établissons les variables de recherche. En deuxième partie, nous présentons la planification de l'expérimentation contrôlée. En troisième partie, nous présentons la méthodologie d'analyse adoptée, en expliquant comment nous avons construit l'oracle et en présentant les mesures de performances appliquées.

Cette section décrit notre expérimentation contrôlée en suivant le cadre de Basili (Basili *et al.*, 1994).

2.1 Définition de l'expérimentation contrôlée

Nous décrivons dans cette section l'expérimentation menée dans le cadre de notre étude. Par la définition de l'objectif, le "quality focus", la perspective, le contexte, les participants et les objets relatifs à notre expérimentation contrôlée. Puis nous définissons nos questions de recherche ainsi que les hypothèses associées et les variables de recherche (les variables indépendantes et dépendantes) de notre étude.

L'*objectif* de cette expérimentation contrôlée consiste (i) à déterminer si la synthèse des éléments de code (classes et méthodes) peut être effectuée à partir de sources d'informations logiciel informelles, telles que le Stack Overflow [1] et les rapports de bogues, plutôt que de n'utiliser que des sources d'informations logiciel formelles telles que le code source et (ii) à déterminer quel type de sources d'informations est le plus pertinent pour la synthèse d'éléments de code.

1. Stack Overflow est un site comportant des questions et des réponses sur un large éventail de sujets reliés à la programmation informatique.

Le *“quality focus”* est la performance des développeurs professionnels mesurée en termes de précision, de rappel et de F-mesure de leurs synthèses par rapport à des synthèses d’annotations des humains. Les synthèses générées par les développeurs professionnels sont produites en consultant plusieurs sources d’informations. Ces dernières sont le (i) code source des éléments de code, (ii) les rapports de bogues liés aux éléments de code, (iii) les publications Stack Overflow liées aux éléments de code et (iiii) la combinaison des trois sources d’informations. La combinaison des sources d’informations consiste à la consultation des trois sources d’informations le code source, le Stack Overflow et les rapports de bogues.

La *perspective* concerne les chercheurs et les praticiens intéressés à (i) connaître les différences entre les différentes documentations logiciel (code source, Stack Overflow et les rapports de bogues) en termes d’aide à la synthèse des éléments de code (classes et méthodes) et (ii) améliorer la précision de leurs outils automatiques.

Le *contexte* implique 24 développeurs professionnels. Ces participants se sont déplacés à notre laboratoire de recherche se trouvant à l’ÉTS afin d’effectuer l’expérimentation contrôlée qui consiste à remplir trois questionnaires (un pré-questionnaire, un questionnaire expérimental et un post-questionnaire). Le questionnaire expérimental est le questionnaire principal où les participants sont invités à synthétiser les éléments de code (classes et méthodes). Les deux autres questionnaires (pré-questionnaire et post-questionnaire) sont composés de questions complémentaires sur les participants ainsi que des informations supplémentaires sur l’expérimentation contrôlée.

Les *participants* sollicités à la l’expérimentation contrôlée sont des développeurs professionnels titulaires de baccalauréat, maîtrise et/ou doctorat. Ces participants sont issus de 16 entreprises et institutions canadiennes dont CGI, Deloitte, Ciena, B-CITI, Fujitsu Conseil Canada, Gestisoft, 360Medlink, Alithya, Banque Nationale du Canada, Rideau Recognition Solutions, GIRO, Ooda Technologies, SecureOps, CAE, PSP Investments et le ministère des Services partagés Canada, branche : Superinformatique.

Les *objets* consistent en huit éléments de code (classes et méthodes) à synthétiser, ils sont extraits aléatoirement de quatre projets open-source *Java* : Eclipse, JMeter, Tomcat et Netbeans. Ces huit éléments de code sont constitués de quatre classes et quatre méthodes. Pour chaque participant, nous présentons quatre éléments de code (deux classes et deux méthodes), en variant les éléments de code et leur ordre de présentation pour chaque participant. Jusqu'à ce que nous récoltions les synthèses de chaque élément de code avec chaque source d'informations étudiée. Nous avons répété cela trois fois, afin de collecter trois synthèses différentes pour chaque élément de code et de chaque source d'informations.

2.1.1 Questions de recherche et formulation des hypothèses

Dans cette expérimentation contrôlée, nous avons posé deux questions de recherche principales et sept questions secondaires qui découlent de la deuxième question de recherche, afin de déterminer comment les développeurs professionnels performent-ils lorsqu'ils utilisent les différents types de sources d'informations logiciel au cours de la tâche de synthèse de code. Contrairement à la première question de recherche, nous avons attribué une hypothèse nulle et une hypothèse alternative à chaque autre question de recherche (principales ou spécifiques).

Les principales questions de recherche adressées dans notre étude sont les suivantes :

- **QR1 :** Quelle est la performance des développeurs professionnels lors de la réalisation des tâches de synthèse des éléments de code quand ils utilisent différents types de sources d'informations logiciel ?
- **QR2 :** Dans quelle mesure l'utilisation de différentes sources d'informations a-t-elle un impact sur la performance des développeurs professionnels lors de la tâche de synthèse des éléments de code ?

La performance des développeurs professionnels est mesurée par la précision, le rappel et la F-mesure des synthèses obtenues par les développeurs professionnels par rapport aux synthèses

établies par les annotateurs impliqués dans la construction de notre oracle (résultats censés être corrects).

La première question **QR1** de recherche vise à déterminer les performances des développeurs professionnels lors de la synthèse des éléments de code avec l'utilisation des différents types de sources d'informations logiciel, c'est-à-dire le code source, le Stack Overflow, les rapports de bogues et la combinaison de ces trois types de sources d'informations logiciel (code source, Stack Overflow et les rapports de bogues).

La deuxième question de recherche **QR2** cherche à définir les différences entre l'utilisation des sources d'informations (code source, rapports de bogues, Stack Overflow ou la combinaison de ces trois types de sources d'informations logiciel) dans la tâche de synthèse en termes des performances des développeurs professionnels. Cette question de recherche concerne également la performance des développeurs professionnels en termes de temps pris pour effectuer les tâches de synthèse en utilisant les différents types de sources d'informations logiciel examinées.

L'hypothèse nulle liée à la deuxième question de recherche **QR2** est formulée comme suit :

- $\mathbf{H_{0,QR2}}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation de différents types de sources d'informations en termes de précision, rappel et F-mesure.*

  L'hypothèse nulle suppose qu'il n'existe pas de différences entre les performances des développeurs professionnels lors de l'utilisation des différentes sources d'informations, c'est-à-dire le code source, le Stack Overflow, les rapports de bogues et la combinaison de ces trois types de sources d'informations (code source, Stack Overflow et les rapports de bogues). La performance est mesurée par la précision, rappel et F-mesure.

L'hypothèse alternative liée à la deuxième question de recherche **QR2** est formulée comme suit :

- $\mathbf{H_{a,QR2}}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation de différents types de sources d'informations en termes de précision, rappel et F-mesure.*

L'hypothèse alternative suppose qu'il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation des différentes sources d'informations, c'est-à-dire code source, rapports de bogues, Stack Overflow et combinaison des trois sources d'informations (code source, Stack Overflow et les rapports de bogues) en termes de précision, rappel et F-mesure.

Les questions de recherche spécifiques qui découlent de la deuxième question de recherche **RQ2** consistent à faire des comparaisons entre les performances des développeurs professionnels lors de l'utilisation des différentes sources d'information considérées. Elles sont formulées avec leurs hypothèses nulles et leurs hypothèses alternatives comme suit :

- **$QR2_1$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du code source versus le Stack Overflow lors de la tâche de synthèse des éléments de code ?

  L'hypothèse nulle de la **$QR2_1$** est formulée comme suit :

  $H_{0,QR2_1}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation du code source versus Stack Overflow en termes de précision, rappel et F-mesure.*

  L'hypothèse alternative de la **$QR2_1$** est formulée comme suit :

  $H_{a,QR2_1}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation du code source versus Stack Overflow en termes de précision, rappel et F-mesure.*

- **$QR2_2$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l' utilisation du code source versus les rapports de bogues lors de la tâche de synthèse des éléments de code ?

  L'hypothèse nulle de la **$QR2_2$** est formulée comme suit :

  $H_{0,QR2_2}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation du code source versus les rapports de bogues en termes de précision, rappel et F-mesure.*

  L'hypothèse alternative de la **$QR2_2$** est formulée comme suit :

$\mathbf{H_{a,QR2_2}}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation du code source versus les rapports de bogues en termes de précision, rappel et F-mesure.*

- **QR2$_3$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l' utilisation du code source versus la combinaison des sources d'informations lors de la tâche de synthèse des éléments de code ?

  L'hypothèse nulle de la **QR2$_3$** est formulée comme suit :

  $\mathbf{H_{0,QR2_3}}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation du code source versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

  L'hypothèse alternative de la **QR2$_3$** est formulée comme suit :

  $\mathbf{H_{a,QR2_3}}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation du code source versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

- **QR2$_4$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l' utilisation du Stack Overflow versus rapports de bogues des sources lors de la tâche de synthèse des éléments de code ?

  L'hypothèse nulle de la **QR2$_4$** est formulée comme suit :

  $\mathbf{H_{0,QR2_4}}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation du Stack Overflow versus rapports de bogues en termes de précision, rappel et F-mesure.*

  L'hypothèse alternative de la **QR2$_4$** est formulée comme suit :

  $\mathbf{H_{a,QR2_4}}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation du Stack Overflow versus rapports de bogues en termes de précision, rappel et F-mesure.*

- **QR2$_5$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l' utilisation du Stack Overflow versus la combinaison des sources d'informations des sources lors de la tâche de synthèse des éléments de code ?

L'hypothèse nulle de la **QR2$_5$** est formulée comme suit :

H$_{0,QR2_5}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation du Stack Overflow versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

L'hypothèse alternative de la **QR2$_5$** est formulée comme suit :

H$_{a,QR2_5}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation du Stack Overflow versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

- **QR2$_6$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation des rapports de bogues versus la combinaison des sources d'informations lors de la tâche de synthèse des éléments de code ?

  L'hypothèse nulle de la **QR2$_6$** est formulée comme suit :

  H$_{0,QR2_6}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation des rapports de bogues versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

  L'hypothèse alternative de la **QR2$_6$** est formulée comme suit :

  H$_{a,QR2_6}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation des rapports de bogues versus la combinaison des sources d'informations en termes de précision, rappel et F-mesure.*

- **QR2$_7$** : Existe-t-il une différence entre le temps moyen consacré par les développeurs professionnels lors des tâches de synthèse des éléments de code en utilisant les différents types de sources d'informations ?

2.1.2 Variables de recherche

Variable indépendante

La principale variable indépendante de cette étude expérimentale est le type de sources d'informations logiciel utilisé au cours de la réalisation des tâches de synthèse par les développeurs professionnels. Il existe quatre différentes valeurs pour ce facteur :

- code source ;
- Stack Overflow ;
- rapports de bogues ;
- combinaison des sources d'informations.

Variable dépendante

La variable dépendante de notre étude est la performance des participants à synthétiser les éléments de code qui est mesurée par l'utilisation des mesures de la précision, rappel et F-mesure appliquées entre les synthèses générées par les développeurs professionnels et un oracle construit par des annotateurs. Puis nous avons utilisé la F-mesure afin (i) d'agréger la précision et le rappel et (ii) pour fournir une mesure globale de la précision et le rappel.

Pour mesurer notre variable dépendante, nous avons construit manuellement l'oracle par consensus de trois annotateurs, où ils ont construit trois oracles préliminaires qui ont mené à la construction d'un seul oracle. Puis, nous comparons les similitudes des mots qui se trouvent dans les synthèses des développeurs avec les mots qui composent l'oracle (la section 2.3.1 fournit plus de détails sur la construction de l'oracle).

2.2 Planification de l'expérimentation contrôlée

La méthodologie adoptée à l'aboutissement de cette étude est la réalisation d'une expérimentation contrôlée. Sa réalisation a nécessité une planification de l'expérimentation contrôlée, par

la définition des points suivants : Définition des projets open-source *Java*, définition des méthodes et classes à synthétiser qui constituent les éléments de code étudiés, la préparation des questionnaires et le recrutement des participants. Nous précisons que le temps de préparation de cette expérimentation a duré cinq mois.

2.2.1 Méthodes et classes

Nous avons choisi aléatoirement un ensemble d'éléments de code, au niveau d'API, d'un pool non probabiliste d'éléments de code, c'est-à-dire, classes et méthodes tirées de quatre projets *Java* open-source (Eclipse, NetBeans, JMeter et Apache Tomcat).

Tableau 2.1 Caractéristiques des projets open-source Java.

Projet	**#[a]Lignes de code**	**#Classes**	**#Méthodes**
Eclipse [b]	2,877,585	16,105	94,172
JMeter [c]	915,991	5,627	33,413
Tomcat [d]	553,874	2,182	13,668
NetBeans [e]	1,522,912	12,139	73,431

a. nombre
b. https://eclipse.org
c. https://netbeans.org
d. https://tomcat.apache.org
e. https://jmeter.apache.org

Nous avons opté pour quatre projets *Java* open-source afin de diversifier et de généraliser les méthodes et classes choisies. Les quatre projets ont été choisis aléatoirement et suivant leurs complexités en termes de nombre de lignes de code, nombre de classes et nombre de méthodes de chaque projet. Ces caractéristiques sont présentées dans le tableau 2.1.

Les éléments de code se composent de quatre classes et de quatre méthodes. Le pool original adhère aux critères suivants :

- Les éléments de code ne doivent pas être triviaux, en d'autres termes, nous ne voulons pas que les éléments de code soient synthétisés sans avoir besoin d'une source d'informations particulière.
- Chaque élément de code dispose d'au moins une page dans Stack Overflow.
- Chaque élément de code dispose d'au moins un rapport de bogues.

Nous avons défini ces caractéristiques par consentement de quatre chercheuses en ce qui concerne la difficulté des classes et méthodes, lors des tests pré-expérimentaux. Ces chercheuses sont composées de la professeure Olga Baysal (Université de Carleton, Ottawa, Canada), professeure Bonita Sharif (Université de Youngstown State, Ohio, États-Unis), Professeure Latifa Guerrouj (Directrice de ce mémoire) et l'auteure de ce mémoire.

Tableau 2.2 Partie 1 : Caractéristiques des éléments de code sélectionnés.

Projet	**Niveau**	**Nom complet**	**Difficulté**
Eclipse 4.2	Méthode1	org.eclipse.core.databinding.Binding.dispose	Difficile
Eclipse 4.2	Classe1	org.eclipse.swt.SWTError	Moyen
JMeter 3.2	Méthode2	org.apache.jmeter.Jmeter.convertSubTree	Moyen
JMeter 3.2	Classe2	org.apache.jmeter.samplers.SampleResult	Moyen
Tomcat 7	Méthode3	org.apache.catalina.realm.JDBCRealm.getRoles	Difficile
Tomcat 7	Classe3	org.apache.catalina.valves.ValveBase	Difficile
NetBeans 7.4	Méthode4	org.netbeans.api.progress.ProgressUtils. runOffEventDispatchThread	Difficile
NetBeans 7.4	Classe4	org.openide.nodes.CHildFactory.Detachable	Difficile

Le nom complet de chaque méthode et classe sélectionnée et leurs niveaux de difficulté sont représentés dans le tableau 2.2. Le niveau de difficulté a été attribué par consensus entre les professeures Olga Baysal (Université de Carleton, Ottawa, Canada), Bonita Sharif (Université de Youngstown State, Ohio, États-Unis) et Latifa Guerrouj (Directrice du présent mémoire) qui sont considérées comme étant habituées et expertes dans le domaine, ainsi que l'auteure de ce mémoire.

À des fins d'alléger le nom de chaque élément de code, nous leurs attribuons un nom tel qu'illustré dans 2.2. Par exemple, la méthode1 est attribuée à la méthode org.eclipse.core.databinding.Binding.dispose, la **classe1** est attribuée à la méthode org.eclipse.swt.SWTError, ainsi de suite (voir tableau 2.2).

Tableau 2.3 Partie 2 : Caractéristiques des éléments de code sélectionnés.

Élément de code	**# Ligne de code**	**# Lignes de commentaires**	**# Pages SO** [a]	**# Pages RB** [b]
Méthode1	18	1	42	37
Classe1	34	88	56	113
Méthode2	51	2	1	3
Classe2	721	449	200	14
Méthode3	42	14	2	1
Classe3	133	104	39	26
Méthode4	14	18	1	41
Classe4	70	159	1	1

a. Stack Overflow
b. Rapports de bogues

Le tableau 2.3 complète le tableau 2.2 en ce qui concerne les caractéristiques des éléments de code sélectionnés. En définissant le nombre de pages Stack Overflow, le nombre de rapports de bogues, le nombre de lignes de code et le nombre de lignes de commentaires pour chaque élément de code.

Telle qu'illustrée dans le tableau 2.3, la documentation peut s'avérer faible en informations. Par exemple, la **méthode2** possède deux lignes de commentaires, une page Stack Overflow et trois pages de rapports de bogues. Cela s'explique par le fait que ces résultats de recherche sont réalistes et reflètent une situation probable. De plus, les éléments de code ont été choisis aléatoirement d'un ensemble non probabiliste. Par conséquent, le nombre de pages d'une recherche Stack Overflow ou des rapports de bogues dans cette expérimentation contrôlée est représentatif d'une recherche réelle.

2.2.2 Tâches expérimentales

Les participants sont invités à synthétiser quatre éléments de code (deux méthodes et deux classes). Ces quatre éléments appartiennent à l'ensemble des huit éléments de code composés de quatre classes et des quatre méthodes (décrites dans le tableau 2.2).

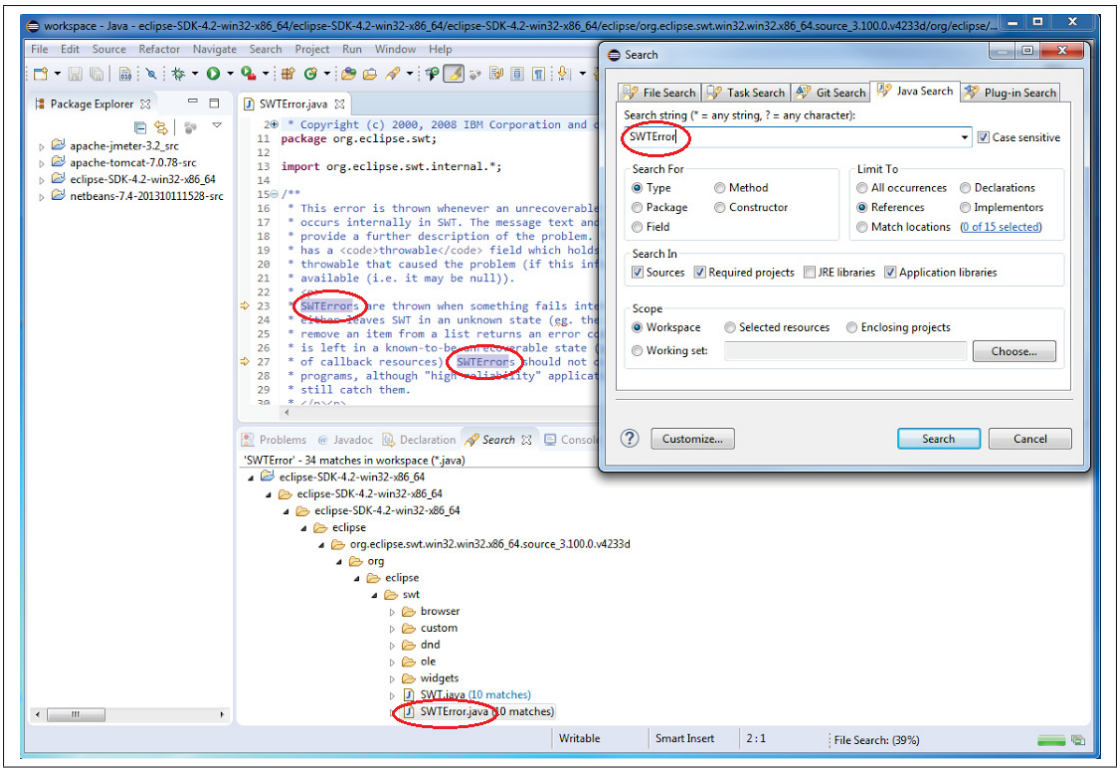


Figure 2.1 Illustration d'une recherche d'un élément de code par l'utilisation du code source.

Chaque élément de code est synthétisé en utilisant une des quatre sources d'informations logiciel suivantes :

- **Code source :** Chaque participant est invité à consulter le code source du projet open-source de la méthode ou la classe donnée à synthétiser. En effectuant une recherche de la classe ou méthode dans l'EDI (Environnement de développement intégré) Eclipse tel que *Eclipse Oxygen*. La figure 2.1 illustre une recherche d'une classe avec l'utilisation du code source, où l'élément de code est la **classe1** *SWTError* du package *org.eclipse.swt*.
- **Stack Overflow :** Lorsqu'il est demandé d'utiliser Stack Overflow afin de synthétiser les éléments de code, un lien vers la recherche Stack Overflow est donné au participant afin qu'il puisse accéder directement aux publications liées à l'élément de code. Prenons comme exemple la recherche de la **classe2** *SampleResult* du package *org.apache.jmeter.samplers.SampleResult* dans Stack Overflow illustrée dans la figure 2.2

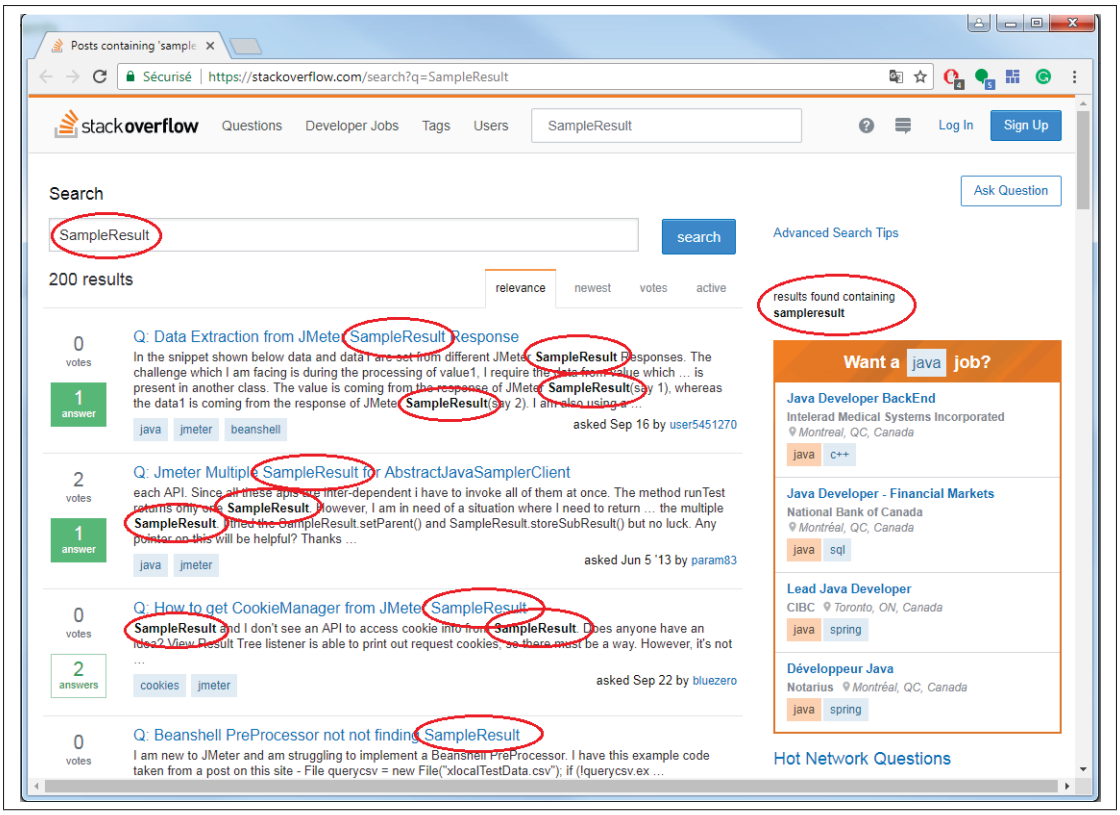


Figure 2.2 Illustration d'une recherche d'un élément de code avec l'utilisation du Stack Overflow.

où il existe 200 publications contenant la **classe2**. Le participant doit lire suffisamment de publication Stack Overflow afin de pouvoir synthétiser correctement les éléments de code.

- **Rapports de bogues :** L'utilisation des rapports de bogues pour la synthèse des éléments de code est illustrée dans la figure 2.3 où le participant est invité à consulter un lien menant à plusieurs rapports de bogues liés à l'élément de code. Dans l'exemple illustré dans la figure 2.3, l'élément de code recherché est la **méthode4** *runOffEventDispatchThread* issue du package *org.netbeans.api.progress.ProgressUtils*. Dans ce cas de figure, il existe 41 rapports de bogues contenant la **méthode4**. Le participant doit lire suffisamment de rapports de bogues afin de pouvoir synthétiser correctement les éléments de code.
- **Combinaison des sources d'informations logiciel (code source, Stack Overflow et les rapports de bogues) :** L'utilisation des trois documentations (code source, Stack Overflow et les rapports de bogues) afin de synthétiser un seul élément de code est une combinaison

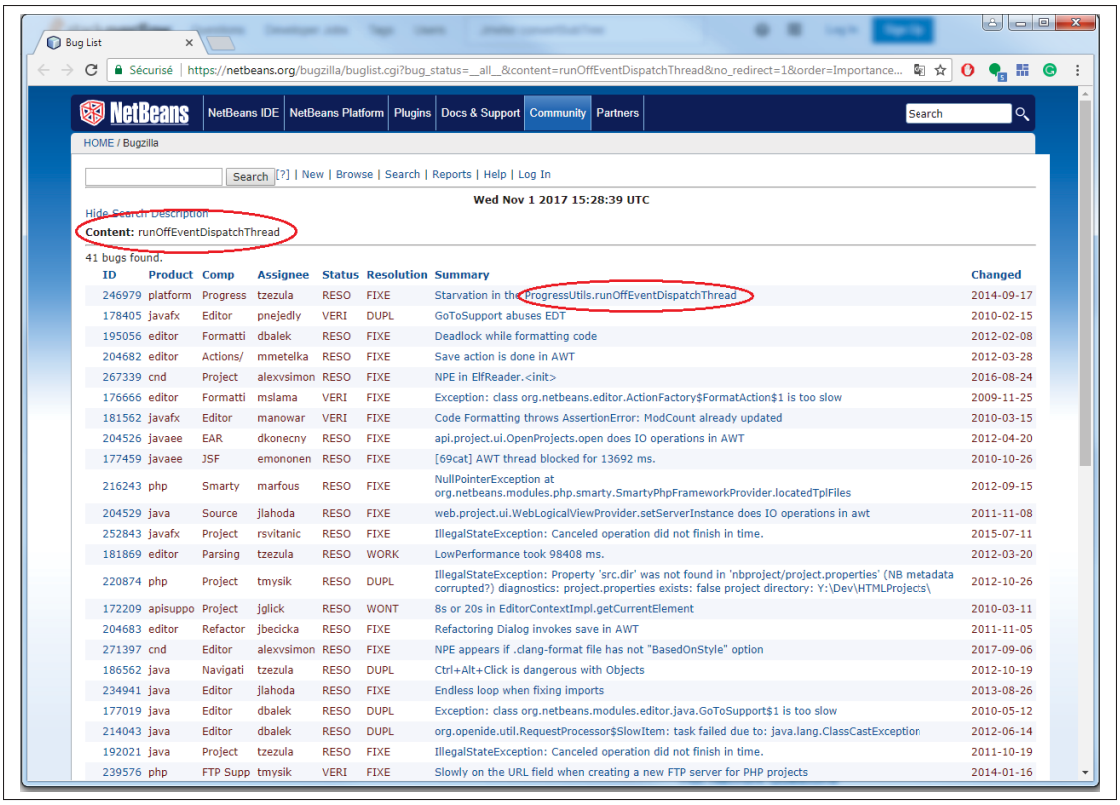


Figure 2.3 Illustration d'une recherche d'un élément de code avec l'utilisation des rapports de bogues.

des trois étapes décrites précédemment et illustrées dans les trois figures 2.1, 2.2 et 2.3. Le participant consulte les documentations dans l'ordre qu'il souhaite. Dès qu'il récolte suffisamment d'informations concernant l'élément de code, il procède à la synthèse de ce dernier.

L'expérimentation menée est une expérimentation contrôlée où le participant se déplace vers le laboratoire de recherche (au sein de l'ÉTS) où se déroule l'expérimentation contrôlée selon ses disponibilités. Le participant avait accès à un micro-ordinateur avec un seul écran, où les questionnaires étaient prêts à être utilisés, ainsi que l'EDI Eclipse avec les projets open-source déjà importés dans l'EDI Eclipse. Afin qu'il puisse chercher facilement le code source des éléments de code.

En général, l'expérimentation contrôlée s'est déroulée avec un participant à la fois. Il y a eu quelques cas (trois cas) où deux à quatre participants ont conduit l'expérimentation contrôlée en même temps. L'expérimentation était contrôlée par la présence de l'auteure de ce mémoire afin de s'assurer du bon déroulement. Le personnel de l'ÉTS a mis à notre disposition cinq postes munis des micro-ordinateurs et dispositions nécessaires dans le même laboratoire de recherche afin de faire face à cette situation.

Nous avons utilisé l'outil *Camtasia Studio* afin d'enregistrer la recherche effectuée par les participants. Cet outil est un logiciel pour créer des tutoriels vidéos et des présentations directement via screencast, ou via un plug-in d'enregistrement direct sur Microsoft PowerPoint. La zone d'écran à enregistrer peut être choisie librement, et les enregistrements audios ou autres enregistrements multimédias peuvent être enregistrés en même temps ou ajoutés séparément de toute autre source et intégrés dans le composant Camtasia du produit.

À chaque participant une séquence de quatre taches est attribuée. Cette dernière est définie dans ce qui suit.

2.2.3 Design de l'expérimentation contrôlée

En génie logiciel empirique, toute étude expérimentale suit un design expérimental spécifique. Dans notre cas, nous avons opté pour un design aléatoire pour l'assignation des sujets, des objets et des traitements (nous insinuons par traitement, la valeur particulière d'un facteur) (Wohlin *et al.*, 2000).

Le tableau 2.4 représente les 24 séquences de tâches expérimentales générées, où chaque ligne représente une séquence et chaque colonne une tâche. Chaque tâche est composée de la source d'informations utilisée (c'est-à-dire, CS = Code Source, SO = Stack Overflow, RB = Rapports de Bogues et Combinaison= Code source, Stack Overflow et rapports de bogues) et de l'élément de code (classe ou méthode) synthétisé.

Tableau 2.4 Design de l'expérimentation contrôlée.

#Participant	Tâche 1	Tâche 2	Tâche 3	Tâche 4
1	RB Classe3	ALL Méthode4	SO Méthode2	CS Classe1
2	RB Classe3	SO Méthode2	ALL Méthode4	CS Classe1
3	SO Méthode2	CS Classe1	RB Classe3	ALL Méthode4
4	SO Classe3	RB Classe2	CS Méthode4	ALL Méthode1
5	ALL Méthode1	RB Classe2	SO Classe3	CS Méthode4
6	RB Classe2	SO Classe3	ALL Méthode1	CS Méthode4
7	SO Classe2	RB Méthode1	CS Méthode3	ALL Classe4
8	RB Méthode1	SO Classe2	CS Méthode3	ALL Classe4
9	SO Classe2	RB Méthode1	ALL Classe4	CS Méthode3
10	SO Méthode1	ALL Classe3	RB Classe1	CS Méthode2
11	ALL Classe3	RB Classe1	SO Méthode1	CS Méthode2
12	ALL Classe3	CS Méthode2	SO Méthode1	RB Classe1
13	CS Méthode1	ALL Classe2	RB Classe4	SO Méthode3
14	SO Méthode3	RB Classe4	CS Méthode1	ALL Classe2
15	SO Méthode3	ALL Classe2	CS Méthode1	RB Classe4
16	ALL Méthode2	CS Classe2	SO Classe4	RB Méthode3
17	RB Méthode3	ALL Méthode2	CS Classe2	SO Classe4
18	ALL Méthode2	RB Méthode3	CS Classe2	SO Classe4
19	CS Classe4	ALL Méthode3	SO Classe1	RB Méthode4
20	RB Méthode4	SO Classe1	CS Classe4	ALL Méthode3
21	CS Classe4	SO Classe1	ALL Méthode3	RB Méthode4
22	CS Classe3	RB Méthode2	SO Méthode4	ALL Classe1
23	ALL Classe1	SO Méthode4	CS Classe3	RB Méthode2
24	ALL Classe1	RB Méthode2	SO Méthode4	CS Classe3

Dans le tableau 2.4 nous décrivons chaque séquence et chaque tâche expérimentale affectée à un participant, c'est-à-dire, l'affectation des objets et des traitements aux participants. En considérant une tâche donnée à un participant comme étant la tâche de synthétiser un élément de code (ClasseX ou méthodeX) en utilisant une documentation (CS = Code Source, SO = Stack Overflow, RB = Rapports de Bogues ou ALL= Code source, Stack Overflow et rapports de bogues). Nous avons généré aléatoirement les séquences à présenter aux participants selon les critères suivants :

- Présenter une séquence par participant, en d'autres termes, un participant passe l'expérimentation une seule fois.

- Présenter quatre différentes tâches par participant (synthétiser avec l'utilisation du code source, synthétiser avec l'utilisation du Stack Overflow, synthétiser avec l'utilisation des rapports de bogues, synthétiser avec l'utilisation d'une combinaison du code source, Stack Overflow, rapports de bogues).
- Générer aléatoirement l'ordre des tâches, dans le but de diversifier l'ordre de présentation des tâches à effectuer.
- Présenter deux classes et deux méthodes par participant.
- Ne pas répéter une classe ou une méthode pour chaque participant. C'est-à-dire le participant ne peut avoir une méthode à synthétiser deux fois ou plus, en utilisant différentes sources d'informations (par exemple, un participant ne peut avoir comme deuxième tâche de synthétiser la **classe2** avec l'utilisation des rapports de bogues puis dans la troisième tâche de synthétiser la **classe2** avec l'utilisation du Stack Overflow).
- Obtenir pour chaque élément de code trois synthèses formulées en utilisant une documentation, c'est-à-dire, obtenir trois synthèses de la méthode3 par l'utilisation du Code source, Stack Overflow, rapports de bogues et la combinaison des trois sources (Code source, Stack Overflow et rapports de bogues) afin d'obtenir un nombre suffisant de synthèses à comparer.
- Couvrir les huit éléments de code (les quatre classes et quatre méthodes).

Pour une meilleure compréhension des séquences, l'**exemple 2.1** illustre une séquence donnée à un participant.

Exemple 2.1. La séquence 10 est composée des quatre tâches suivantes :

- Tâche 1 : L'utilisation du SO (Stack Overflow) afin de synthétiser la Méthode1.
- Tâche 2 : L'utilisation de ALL (la combinaison du code source, rapports de bogues et Stack Overflow) afin de synthétiser la **Classe3**.
- Tâche 3 : L'utilisation des RB (rapports de bogues) afin de synthétiser la **Classe1**.
- Tâche 4 : L'utilisation du CS (code source) afin de synthétiser la méthode2.

2.2.4 Questionnaire expérimental

Tableau 2.5 Résumé du questionnaire expérimental.

ID	Tâche	Directives
T1	Please summarize the method : **org.eclipse.core.databinding.Binding .dispose** using Stack Overflow	1- Open link above. 2- Search the method (**dispose**) in Stack Overflow, while considering the context (**org.eclipse.core.databinding.Binding**). 3- Summarize in a very concise and brief way the given method.
T2	Please summarize the class : **org.apache.catalina.valves.ValveBase** using source code, bug repots, and Stack Overflow	1- Open Eclipse IDE. 2- Search the class (**ValveBase**). 3- Open the source code of the class. 4- Open the link. 5- Search the class (**ValveBase**) in Stack Overflow, while considering the context (**org.apache.catalina.valves**). 6- Open the link. 7- Search the class (**ValveBase**) in Bug reports, while considering the context (**org.apache.catalina.valves**). 8- Summarize in a very concise and brief way the given class.
T3	Please summarize the class : **org.eclipse.swt.SWTError** using bug reports	1- Open the link. 2- Search the class (**SWTError**) in the bug reports, while considering the context (**org.eclipse.swt**). 3- Summarize in a very concise and brief way the given class.
T4	Please summarize the method : **org.apache.jmeter.Jmeter. convertSubTree** using source code	1- Open Eclipse IDE. 2- Search the method (**convertSubTree**). 3- Open the source code of the method. 4- Summarize in a very concise and brief way the given method.

Chaque séquence de tâches expérimentales est présentée au participant sous la forme d'un questionnaire, que nous avons nommé **questionnaire expérimental**. Ce questionnaire expérimental est composé de quatre sections de questions ouvertes qui se succèdent, qui représentent les quatre tâches à effectuer. Les tâches sont formulées en anglais. Chaque tâche se finalise

après que le participant termine de synthétiser l'élément de code (classe ou méthode). En général, les principales étapes à suivre pour chaque tâche sont représentées sous la forme suivante :

- **Synthétiser avec l'utilisation du Stack Overflow**. Il est demandé de suivre les étapes suivantes :
  a. Ouvrir un lien donné qui mène vers Stack Overflow.
  b. Rechercher la méthode ou la classe dans Stack Overflow, en tenant compte du contexte (package).
  c. Synthétiser de manière très concise et brève la méthode ou la classe donnée.

  La première tâche du tableau 2.5 présente les directives et tâches demandées lors de la synthèse d'une **méthode1** utilisant le Stack Overflow par le biais du questionnaire expérimental de la séquence 10 (**exemple 2.1**).
- **Synthétiser avec l'utilisation des trois documentations (code source, Stack Overflow, rapports de bogues)**. Il est demandé de suivre les étapes suivantes :
  a. Ouvrir *Eclipse IDE*.
  b. Rechercher la méthode ou la classe.
  c. Ouvrir le code source de la méthode ou la classe.
  d. Ouvrir un lien donné qui mène vers Stack Overflow.
  e. Rechercher la méthode ou la classe dans Stack Overflow, en tenant compte du contexte (package).
  f. Ouvrir le lien qui mène vers les rapports de bogues.
  g. Rechercher la classe ou la méthode dans les rapports de bogues, en considérant le contexte (package).
  h. Synthétiser de manière très concise et brève la classe ou la méthode donnée.

  La deuxième tâche du tableau 2.5 présente les directives et tâches demandées lors de la synthèse d'une **classe3** utilisant la combinaison des sources d'informations par le biais du questionnaire expérimental de la séquence 10 (**exemple 2.1**).

- **Synthétiser avec l'utilisation des rapports de bogues**, où il est demandé de suivre les étapes suivantes :

  a. Ouvrir le lien qui mène vers les rapports de bogues.

  b. Rechercher la classe ou la méthode dans les rapports de bogues, en considérant le contexte (package).

  c. Synthétiser de manière très concise et brève la classe ou la méthode donnée.

  La troisième tâche du tableau 2.5 présente les directives et tâches demandées lors de la synthèse d'une **classe1** utilisant des rapports de bogues par le biais du questionnaire expérimental de la séquence 10 (**exemple 2.1**).

- **Synthétiser avec l'utilisation du code source**. Il est demandé de suivre les étapes suivantes :

  a. Ouvrir *Eclipse IDE*.

  b. Rechercher la méthode ou la classe.

  c. Ouvrir le code source de la méthode ou la classe.

  d. Synthétiser de manière très concise et brève la méthode ou la classe donnée.

  La quatrième tâche du tableau 2.5 présente les directives et tâches demandées lors de la synthèse d'une **méthode2** utilisant le code source par le biais du questionnaire expérimental de la séquence 10 (**exemple 2.1**).

Le tableau I-1 de l'**annexe I** présente les 24 liens, où chacun d'eux mène vers un questionnaire expérimental lié à une séquence. Les questionnaires ont été réalisés avec l'utilisation de *Google Forms*.

Dans l'annexe II, nous avons illustré le questionnaire expérimental, afin d'avoir une idée sur la structure exacte et la présentation du questionnaire expérimental.

Contexte des éléments de code

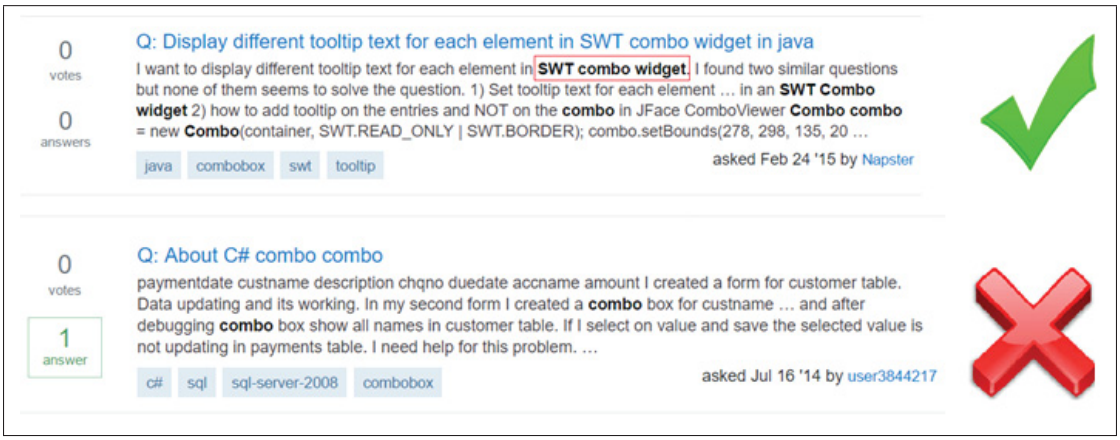


Figure 2.4 Illustration d'une publication Stack Overflow dont le contexte est valide.

Les participants synthétisent les éléments de code (classe ou méthode) en tenant compte du contexte. Le contexte d'un élément de code est le lien qu'une classe ou d'une méthode possède avec le package dont elle est issue. Par exemple, le nom complet d'une classe donnée est *org.eclipse.swt.widgets.Combo*. Le participant est dans l'obligation de chercher la classe *Combo* qui est liée au package *org.eclipse.swt.widgets* dans les différentes publications Stack Overflow et rapports de bogues. Pour une meilleure compréhension de la notion du contexte, la figure 2.4 illustre les choix valides et non valides dont le participant aura à distinguer.

Exemples de synthèses obtenues

À travers le questionnaire expérimental, chaque participant a fourni quatre synthèses. Au final, nous avons analysé 96 synthèses.

Afin de donner une idée sur le type de synthèses d'élément de code obtenues par les développeurs professionnels, nous présentons dans le tableau 2.6 les synthèses obtenues pour les

Tableau 2.6 Exemples de synthèses de quatre éléments de code fournies par les développeurs professionnels.

Source d'informations	**Élément de code**	**Synthèse par développeur**
Code source	Classe4	"This class is a kind of generic observer for Child Factory derived classes, it is used to notify the first use or change of a target class, and also the end-life / dispose of that class."
Stack Overflow	Classe3	"ValveBase works as a gateway for APIs. It is in charge of validation of api requests. It'is used to filter requests and making necessary redirection."
Rapports de bogues	Méthode3	"The method grant access based on the role given to an user/program, for querying a DB. The method needs to make authentication to authorize the caller."
Combinaison	Méthode4	"Receives a Runnable object. Opens a blocking UI with a progress bar. Runs the Runnable Object. Cancel option appears if operation takes too long to complete (10 seconds)."

classes (**classe3** et **classe4**) et les méthodes (**méthode3** et **méthode4**) avec l'utilisation du code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations.

2.2.5 Pré-questionnaire

Avant d'accomplir leur tâche de synthèse des éléments de code, les participants ont répondu à un pré-questionnaire afin d'obtenir des informations intéressantes les concernant. Le pré-questionnaire se compose d'une identification du participant, neuf questions fermées et deux questions semi-fermées.

Tableau 2.7 Résumé du pré-questionnaire.

ID	**Questions**
Question 1	Gender
Question 2	The age range
Question 3	How many years of active programming experience do you have ?
Question 4	What is your level of expertise in the *Java* programming language ?
Question 5	Please select all the degrees you have and are currently enrolled in.
Question 6	Current Positions (student, working in industry, etc.)
Question 7	How many years of work experience do you have in industry ?
Question 8	Do you use Stack Overflow to find solutions to your coding problems ?
Question 9	Do you use/read bug reports to find solutions to issues while coding ?
Question 10	Have you contributed (code and/or documentation) to an open source project ?
Question 11	Which of the following IDEs are you familiar with ? (By familiar we mean you are able to work in fairly well).

Les questions posées aux participants dans le pré-questionnaire sont formulées en anglais. Le tableau 2.7 illustre le résumé des 11 questions posées.

Les informations récoltées du pré-questionnaire sont les suivantes :

- Leurs sexes (Femme ou Homme).
- Leurs tranches d'âge (<18 années, 18 - 25 ans, 26 - 30 ans, 31 - 35 ans, 36 - 40 ans, 41 - 45 ans, 46 - 50 ans ou > 50 ans).
- Le nombre d'années d'expérience en programmation (<1 année, Entre 1 et 2 ans, Entre 3 et 5 ans, Entre 6 et 10 ans ou > 10 ans).
- Leurs niveaux d'expertise dans le langage de programmation *Java* (Faible, intermédiaire, Bon, Très bon, excellent).
- Leurs niveaux d'études (baccalauréat, maîtrise, doctorat et/ou Autre).

- Leurs postes actuels (travaille actuellement dans l'industrie, travaille actuellement dans un milieu universitaire, actuellement étudiant, actuellement membre du corps professoral ou actuellement en post-doctoral)
- Leurs nombres d'années d'expérience de travail en industrie (Aucun <1 année, Entre 1 et 2 ans, Entre 3 et 5 ans, Entre 6 et 10 ans ou > 10 ans).
- S'ils utilisent le Stack Overflow pour trouver des solutions à leurs problèmes de codage (Oui ou Non).
- S'ils utilisent les rapports de bogues pour trouver des solutions à leurs problèmes de codage (Oui ou Non).
- S'ils ont contribué (code et/ou documentation) à un projet open-source (Oui ou Non).
- Les IDEs qu'ils connaissent ou utilisent le plus souvent (Eclipse, Visual Studio, Netbeans, IntelliJ et/ou Autre).

Comme pour le questionnaire expérimental, le pré-questionnaire a été réalisé en utilisant *Google Forms*. Dans l'annexe II, nous avons illustré le pré-questionnaire, afin d'avoir une idée sur la structure exacte et la présentation du pré-questionnaire.

2.2.6 Post-questionnaire

Après avoir fini de remplir le pré-questionnaire et le questionnaire expérimental (suivant cet ordre), les participants sont invités à répondre à un post-questionnaire afin d'obtenir des informations supplémentaires sur les données recueillies au cours de l'expérimentation contrôlée.

Les questions posées aux participants dans le post-questionnaire sont formulées en anglais. Le tableau 2.8 illustre une synthèse des huit questions posées en anglais. Le post-questionnaire est constitué d'une identification du participant, de sept questions fermées et d'une seule question ouverte (tableau 2.8)

Les informations récoltées du pré-questionnaire sont les suivantes :

- Leurs connaissances des projets open-source (Eclipse, NerBeans, JMeter et Tomcat). Pour savoir si ces connaissances les ont aidées dans leurs réponses. Pour chaque projet open-

Tableau 2.8 Résumé du post-questionnaire.

ID	Questions
Question 1	Were you familiar with the code of the following projects or parts of these projects before this study ?
Question 2	Rate the usefulness of each type of information with respect to how helpful they were to summarize the API elements in the study.
Question 3	Rate the usefulness of the different types of contexts present in **source code** that helped you to summarize the API elements.
Question 4	Rate the usefulness of the different types of contexts present in **Stack Overflow (SO) documents** that helped you to summarize the API elements
Question 5	Rate the usefulness of the different types of contexts present in **bug reports** that helped you to summarize the API elements
Question 6	Overall, how difficult did you find the study ?
Question 7	Overall, do you feel you spent sufficient time to summarize the API elements ?
Question 8	Comments

source, le participant a noté son niveau de connaissance entre : Extrêmement familier, Modérément familier, Quelque peu familier, Légèrement familier ou Pas du tout familier.

- L'utilité de chaque type d'informations (Code source, Stack Overflow, rapports de bogues et la combinaison des trois sources d'informations) lors de la synthèse des éléments de code dans l'expérimentation contrôlée. En les considérant comme étant : Extrêmement utile, Modérément utile, Quelque peu utile, Légèrement utile ou Pas du tout utile.
- L'utilité des différents types de contextes présents dans le code source qui les ont aidés pour synthétiser les éléments de code, tels que les commentaires (ligne et commentaires), les Noms des identifiants, les lignes de code, la lisibilité générale du code (par exemple, bonne indentation, structure, etc.). En les considérant comme étant : Extrêmement utile, Modérément utile, Quelque peu utile, Légèrement utile ou Pas du tout utile.
- L'utilité des différents types de contextes présents dans le Stack Overflow qui les ont aidés à synthétiser les éléments de code, tels que les exemples de code, les commentaires des

utilisateurs, les empiler des traces, les descriptions textuelles (autre que le code exemples et pile traces), les votes, les tags des questions, la réputation et/ou le profil de l'utilisateur, la date d'un commentaire ou d'une réponse.

- L'utilité des différents types de contextes présents dans les rapports de bogues qui les ont aidés à synthétiser les éléments de code, les exemples de code, les commentaires des utilisateurs, les correctifs proposés, les cas de test, la description textuelle des rapports de bogues. En les considérant comme étant : Extrêmement utile, Modérément utile, Quelque peu utile, Légèrement utile ou Pas du tout utile.
- La difficulté de l'expérimentation contrôlée, en la notant sur une échelle de Likert, de plus facile (1) à plus difficile (5).
- L'estimation du temps consacré à l'expérimentation, en la notant sur une échelle de Likert, de peu de temps (1) à suffisamment de temps (5).
- Laisser un commentaire qui concerne l'expérimentation contrôlée.

Comme pour le questionnaire expérimental et le pré-questionnaire, le post-questionnaire a été réalisé en utilisant *Google Forms*. Dans l'annexe II, nous avons illustré le post-questionnaire, afin d'avoir une idée sur la structure exacte et la présentation du post-questionnaire.

Participants

Nous avons réalisé une expérimentation contrôlée avec la participation de 24 participants.

Ces 24 participants sont des développeurs professionnels travaillant dans l'industrie. Ils sont issus de 16 entreprises et institutions canadiennes, dont CGI, Deloitte, la société Ciena, B-CITI, Fujitsu Conseil Canada, Gestisoft, 360Medlink, Alithya, Banque Nationale du Canada, Rideau Recognition Solutions, GIRO, Ooda Technologies, SecureOps, CAE, PSP Investments et ministère des Services partagés Canada, branche : Superinformatique.

Nous avons collecté des informations sur tous les participants de l'expérimentation contrôlée en utilisant un pré-questionnaire dans lequel nous demandions aux participants d'auto-évaluer leur niveau de connaissance en programmation *Java* mais aussi de clarifier les années de travail

Tableau 2.9 Caractéristiques des participants.

Caractéristique	**Niveau**	**# de Participants**
Programme d'études	Baccalauréat	3
	Maîtrise	18
	Ph.D.	3
# D'années d'expérience en programmation	Entre 1 & 2 années	6
	Entre 3 & 5 années	10
	Entre 6 & 10 années	5
	> 10 années	3
# D'années d'expérience en industrie	< 1 année	2
	Entre 1 & 2 années	8
	Entre 3 & 5 années	4
	Entre 6 & 10 années	7
	> 10 années	3
Expertise en *Java*	Faible	3
	Passable	5
	Bon	10
	Très bon	6
Utilisateur SO	Oui	21
	Non	3
Utilisateur RB	Oui	15
	Non	9
Contribution projets open-source	Oui	9
	Non	15

dans l'industrie, leur niveau d'éducation, leur contribution dans un projet open-source, leur utilisation des rapports de bogues ainsi que Stack Overflow (tel qu'expliqué dans la section 2.2.5).

Les informations collectées indiquent qu'en plus d'être des développeurs professionnels, ils ont au moins un niveau d'étude de baccalauréat. Autre information, tous les participants ont au moins un an d'expérience en programmation dans le secteur en tant que développeur professionnel. Le tableau 2.9 résume les principales caractéristiques des participants (recueillies à partir du pré-questionnaire) des 24 développeurs professionnels impliqués dans notre étude.

Les éléments de code présentés aux participants ont été choisis suivant les séquences décrites dans le tableau 2.4. Chaque participant est invité à synthétiser quatre éléments de code (deux

méthodes et deux classes) à travers sa lecture des sources d'informations logiciel, dont le code source, le Stack Overflow, les rapports de bogues et la combinaison des trois sources (code source, le Stack Overflow et les rapports de bogues).

Avant de passer l'expérimentation contrôlée, les participants devaient signer un consentement concernant leur participation. Le consentement consiste en un formulaire décrivant l'étique ainsi que les objectifs de l'expérimentation et son déroulement. Aucune hypothèse de recherche n'a été divulguée. Nous avons soumis une demande de certificat d'étique au comité d'éthique de l'ÉTS avant le déroulement de l'expérimentation de trois mois.

Ce formulaire de consentement vise à expliquer en détail au participant le déroulement de l'expérimentation contrôlée ainsi que son consentement à utiliser ses informations recueillies de l'expérimentation. Le formulaire de consentement est un formulaire fourni par l'ÉTS afin de respecter l'étique de l'expérimentation contrôlée.

2.3 Méthodes d'analyse

Nos statistiques ont été analysées en utilisant le langage et l'environnement dédié aux calculs statistiques *R* qui est dédié aux statistiques et à la science des données. Nous avons utilisé l'implémentation logiciel *GNU R* de la version 3.4.0. Les procédures statistiques utilisées pour analyser nos résultats sont décrites dans les sections suivantes.

2.3.1 Construction de l'Oracle

Pour évaluer les performances des développeurs professionnels en termes de précision, rappel et F-mesure, et les comparer lors de l'utilisation des sources d'informations, nous avons construit un Oracle, c'est-à-dire, l'ensemble des synthèses des éléments de code qui sont censées être correctes et concises. L'oracle contient des synthèses du but des éléments de code mais aussi de leur usage.

Nous l'avons construit de façon très rigoureuse suivant une approche consensuelle où un groupe de travail composé de trois collaborateurs qui ont contribué à la construction de l'oracle, dont les professeures Olga Baysal (Université de Carleton, Ottawa, Canada), Bonita Sharif (Université de Youngstown State, Ohio, États-Unis) que nous considérons expertes et familiarisées avec la tâche de synthèse des éléments de code, et l'auteure de ce mémoire. Nous avons procédé de la façon suivante : (i) chacun des trois annotateurs a construit de façon indépendante une synthèse pour chaque élément de code faisant partie des objets de cette expérimentation, (ii) nous nous sommes entendues sur une seule synthèse par élément de code, (iii) pour tous les cas où il y a eu un désaccord concernant les synthèses obtenues une discussion a eu lieu entre deux annotateurs puis un consensus a été atteint.

L'analyse manuelle qui rentre dans le cadre de la construction de l'oracle a été effectuée en se basant sur un nombre très varié de sources d'informations des projets auxquels appartiennent les éléments de code étudiés en considérant non seulement le code source, le Stack Overflow et les rapports de bogues mais aussi la documentation officielle des éléments de code. Ainsi, nous avons construit notre benchmark de synthèses d'annotateurs humains, que nous nommons l'oracle.

Cette approche de construction de l'oracle est une réplique des approches définies par la communauté du génie logiciel empirique, telle que les travaux menés par Haiduc *et al.* (2010). Nous avons aussi compté sur la supervision des professeures Olga Baysal (Université de Carleton, Ottawa, Canada), Bonita Sharif (Université de Youngstown State, Ohio, États-Unis) et Latifa Guerrouj (Directrice du présent mémoire) qui sont considérées comme étant habituées et expertes dans le domaine.

2.3.2 Mesures des performances

La précision, le rappel et la F-mesure sont trois mesures de récupération et de compréhension des informations largement utilisées qui ont été utilisées pour mesurer la performance des mesures logicielles pour diverses tâches de maintenance (Újházi *et al.*, 2010; Ricca *et al.*,

2010). Dans notre étude, nous mesurons la performance des développeurs professionnels lors de la synthèse des éléments de code. Nous expliquons ces mesures dans ce qui suit :

Précision et rappel

Nous avons mesuré les performances des développeurs professionnels par l'utilisation des procédures statistiques : précision, rappel et F-mesure. Étant donné un élément de code classe c_i ou méthode m_i à synthétiser, nous générons les deux ensembles suivants :
$synthese\_annotateur_i = \{oracleTerme_{i,1}, \ldots, oracleTerme_{i,m}\}$
$synthese\_developpeur_i = \{terme_{i,1}, \ldots, terme_{i,n}\}$

Où les termes $oracleTerme_{i,m}$ de l'ensemble *synthese_annotateur* sont les termes de l'oracle fourni par trois annotateurs humains, et les termes $terme_{i,n}$ de l'ensemble *synthese_developpeur* sont les termes qui appartiennent aux synthèses générées par les développeurs professionnels. Sur la base de ces deux ensembles de synthèses, la précision et le rappel sont calculés comme suit :

$$precision_i = \frac{|synthese\_developpeur_i \cap synthese\_annotateur_i|}{|synthese\_developpeur_i|}$$

$$rappel_i = \frac{|synthese\_developpeur_i \cap synthese\_annotateur_i|}{|synthese\_annotateur_i|}$$

F-mesure

Pour fournir une mesure agrégée de la précision et du rappel, nous utilisons la F-mesure, qui est la moyenne harmonique de la précision et du rappel. Son équation est la suivante :

$$\mathrm{F-mesure_i} = \frac{2 \cdot precision_i \cdot rappel_i}{precision_i + rappel_i}$$

2.3.3 Diagramme en boîte à moustaches

Le diagramme en boîte à moustaches est une représentation graphique des données statistiques permettant de visualiser plusieurs paramètres de la distribution d'une variable donnée, tels que le minimum, le premier quartile, la médiane, la moyenne, le troisième quartile et le maximum d'une mesure statistique (McGill *et al.*, 1978).

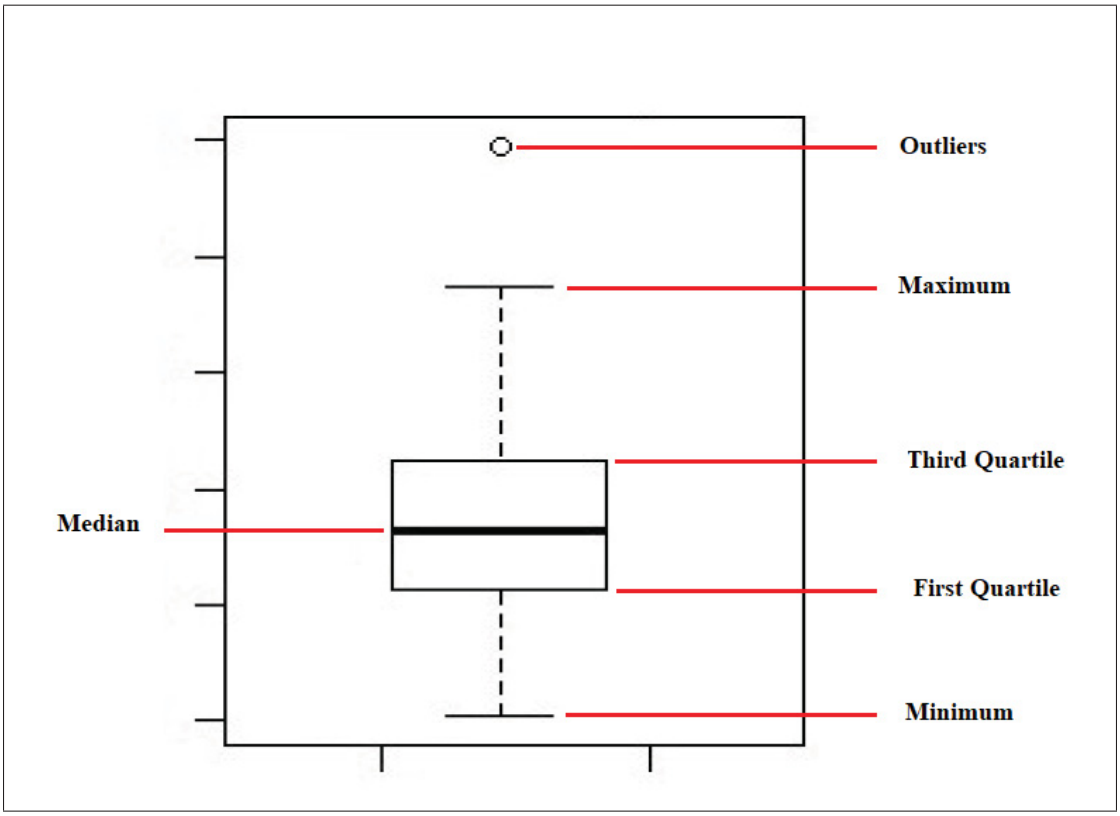


Figure 2.5 Représentation des distributions de données avec les boîtes à moustaches.

La figure 2.5 représente les distributions des données avec l'utilisation des boîtes à moustaches où sont illustrées la médiane, le maximum, le minimum, le premier quartile, le troisième quartile et les données aberrantes.

Dans notre étude, nous nous intéressons à la visualisation des diagrammes en boîte à moustaches de la précision, du rappel et de la F-mesure des synthèses obtenues des participants par

rapport à l'oracle, selon la source d'informations logiciel utilisée (code source, Stack Overflow, rapport de bogues et la combinaison des sources d'informations).

2.3.4 Test statistique : Test de Wilcoxon

Pour comparer la précision, le rappel et la F-mesure des synthèses fournies par les développeurs professionnels et les annotateurs humains, nous avons utilisé un test non paramétrique, qui est le test apparié de Wilcoxon (notre échantillon était petit et les données non distribuées normalement), ce qui indique si la différence médiane entre deux approches est significativement différente de zéro (Wohlin *et al.*, 2000).

2.3.5 Test de multiples corrections des p-values : Correction d'Holm

La p-value est une variable aléatoire dérivée de la distribution des tests statistiques utilisée pour analyser un ensemble de données et pour tester une hypothèse nulle (Hung *et al.*, 1997).

En ce qui concerne la correction de Holm, elle représente un ajustement des p-values lorsque plusieurs tests statistiques dépendants ou indépendants sont effectués simultanément sur un seul ensemble de données (Holm, 1979).

Puisque nous avons appliqué le test de Wilcoxon plusieurs fois, nous avons dû ajuster les p-values en utilisant la procédure de correction de Holm (Holm, 1979).

Afin d'appliquer la correction de Holm, il est nécessaire de suivre les étapes suivantes :

- Les p-values obtenues à partir de plusieurs tests sont classées par ordre croissant.
- La première p-value est multipliée par le nombre k de tests effectués et est considérée comme significative si elle est inférieure à 0.05.
- La seconde p-values est multipliée par $k-1$, ainsi de suite.

2.3.6 Mesure de l'effect-size : Cliff delta (d)

Nous utilisons la mesure non-paramétrique de l'effect-size qui est le Cliff delta (d). Le Cliff delta (d) est défini comme la probabilité qu'un élément d'un échantillon choisi aléatoirement ait une réponse plus élevée qu'un élément issu d'un second échantillon, moins la probabilité inverse (Grissom & Kim, 2005).

Le Cliff delta (d) varie dans l'intervalle $[-1,1]$ et est considéré petit pour $0.148 \leq d < 0.33$, moyen pour $0.33 \leq d < 0.474$ et grand pour $d \geq 0.474$ (Grissom & Kim, 2005).

Les mesures que nous avons utilisées sont les plus répandues dans ce type d'études et domaines connexes, par exemple, les travaux menés par Rastkar *et al.* (2010). Elles reposent sur des comparaisons entre les termes faisant partie des synthèses des développeurs professionnels et ceux apparaissant dans les synthèses des annotateurs humains.

CHAPITRE 3

RÉSULTATS ET DISCUSSIONS

Dans ce chapitre, nous présentons pour chaque question de recherche les résultats obtenus de l'analyse des données récoltées lors de l'expérimentation contrôlée menée dans cette étude. Ensuite, nous identifions les obstacles à la validité de notre expérimentation contrôlée, dont la validité externe, la validité interne, la validité de construction et la validité de conclusion. Enfin, nous terminons par la précision des limites de notre approche.

3.1 QR1 : Quelle est la performance des développeurs professionnels lors des tâches de synthèse des éléments de code quand ils utilisent différents types de sources d'informations ?

Afin d'obtenir une vue globale sur les performances des développeurs professionnels en termes de précision, rappel et F-mesure selon la source d'informations utilisée, nous avons calculé les statistiques descriptives, c'est-à-dire, le minimum, le premier quartile, la médiane, la moyenne, le troisième quartile et le maximum de chaque mesure statistique, et cela pour chaque type de sources d'informations étudiées, c'est-à-dire, l'utilisation du code source, Stack Overflow, rapports de bogues et la combinaison des trois sources d'informations (codes source, Stack Overflow et rapport de bogues).

3.1.1 Performances des développeurs professionnels avec l'utilisation du code source

Tableau 3.1 Performances des développeurs professionnels avec l'utilisation du **code source** lors de la synthèse des éléments de code en termes de précision, rappel et F-mesure.

Mesure	Min	1er Q.	Médiane	Moyenne	3eme Q.	Max
Précision	10.00	14.00	17.47	25.67	26.71	90.91
Rappel	5.56	21.43	35.42	38.35	50.00	71.43
F-mesure	7.14	17.85	23.06	28.80	30.75	80.00

Le tableau 3.1 rapporte les statistiques descriptives, dont la précision, le rappel et la F-mesure calculés pour la comparaison des synthèses de l'oracle avec les synthèses des développeurs professionnels par l'utilisation du code source.

Tel que nous pouvons le constater, les développeurs professionnels atteignent en moyenne 25.67 de précision. Le minimum de précision est de 10 alors que le maximum de précision obtenu est de 90.91.

En ce qui concerne le rappel, les développeurs professionnels atteignent en moyenne 38.35 de rappel. Le minimum de rappel est de 5.56 alors que le maximum de rappel obtenu est de 71.43.

En termes de F-mesure, les développeurs professionnels atteignent en moyenne 28.80 de F-mesure. Le minimum de F-mesure est de 7.14 alors que le maximum de F-mesure obtenu est de 80.00.

Les autres statistiques descriptives y compris le premier quartile, la médiane, et le troisième quartile sont également reportés au niveau du tableau 3.1.

3.1.2 Performances des développeurs professionnels avec l'utilisation du Stack Overflow

Tableau 3.2 Performances des développeurs professionnels avec l'utilisation du **Stack Overflow** lors de la synthèse des éléments de code en termes de précision, rappel et F-mesure.

Mesure	**Min**	**1er Q.**	**Médiane**	**Moyenne**	**3eme Q.**	**Max**
Précision	10.17	15.38	21.58	27.09	33.33	71.43
Rappel	14.29	22.22	25.00	32.63	42.11	75.00
F-mesure	15.87	19.02	21.92	26.77	32.29	51.85

Le tableau 3.2 rapporte les statistiques descriptives dont la précision, le rappel et la F-mesure calculés pour la comparaison des synthèses de l'oracle avec les synthèses des développeurs professionnels par l'utilisation du Stack Overflow.

Tel que nous pouvons le constater, les développeurs professionnels atteignent en moyenne 27.09 de précision. Le minimum de précision est de 10.17 alors que le maximum de précision obtenu est de 71.43.

En ce qui concerne le rappel, les développeurs professionnels atteignent en moyenne 32.63 de rappel. Le minimum de rappel est de 14.29 alors que le maximum de rappel obtenu est de 75.00.

En termes de F-mesure, les développeurs professionnels atteignent en moyenne 26.77 de F-mesure. Le minimum de F-mesure est de 15.87 alors que le maximum de F-mesure obtenu est de 51.85.

Le premier quartile, la médiane, et le troisième quartile sont également reportées au niveau du tableau 3.2.

3.1.3 Performances des développeurs professionnels avec l'utilisation les rapports de bogues

Tableau 3.3 Performances des développeurs professionnels avec l'utilisation des **rapports de bogues** lors de la synthèse des éléments de code en termes de précision, rappel et F-mesure.

Mesure	**Min**	**1er Q.**	**Médiane**	**Moyenne**	**3eme Q.**	**Max**
Précision	8.00	11.43	15.19	20.78	28.79	57.14
Rappel	10.53	18.23	21.83	21.53	25.42	33.33
F-mesure	10.26	14.63	16.80	18.98	22.55	35.29

Le tableau 3.3 rapporte les statistiques descriptives, dont la précision, le rappel et la F-mesure calculés pour la comparaison des synthèses de l'oracle avec les synthèses des développeurs professionnels par l'utilisation des rapports de bogues.

Tel que nous pouvons le constater, les développeurs professionnels atteignent en moyenne 20.78 de précision. Le minimum de précision est de 8 alors que le maximum de précision obtenu est de 57.14.

En ce qui concerne le rappel, les développeurs professionnels atteignent en moyenne 21.53 de rappel. Le minimum de rappel est de 10.53 alors que le maximum de rappel obtenu est de 33.33.

En termes de F-mesure, les développeurs professionnels atteignent en moyenne 18.98 de F-mesure. Le minimum de F-mesure est de 10.26 alors que le maximum de F-mesure obtenu est de 35.29.

Le reste des statistiques descriptives comme le premier quartile, la médiane, et le troisième quartile sont également reportées au niveau du tableau 3.3.

3.1.4 Performances des développeurs professionnels avec l'utilisation de la combinaison des sources d'informations logiciel

Tableau 3.4 Performances des développeurs professionnels avec l'utilisation de la **combinaison des sources d'informations** lors de la synthèse des éléments de code en termes de précision, rappel et F-mesure.

Mesure	**Min**	**1er Q.**	**Médiane**	**Moyenne**	**3eme Q.**	**Max**
Précision	4.17	8.90	17.47	20.16	27.27	54.55
Rappel	6.25	16.67	38.75	34.31	50.66	58.33
F-mesure	5.00	13.34	22.95	23.01	30.89	46.15

Le tableau 3.4 rapporte les statistiques descriptives dont la précision, le rappel et la F-mesure calculés pour la comparaison des synthèses de l'oracle avec les synthèses des développeurs professionnels par l'utilisation de la combinaison des sources d'information, c'est-à-dire, l'utilisation du code source, du Stack Overflow et des rapports de bogues.

Tel que nous pouvons le constater, les développeurs professionnels atteignent en moyenne 20.16 de précision. Le minimum de précision est de 4.17 alors que le maximum de précision obtenu est de 54.55.

En ce qui concerne le rappel, les développeurs professionnels atteignent en moyenne 34.31 de rappel. Le minimum de rappel est de 6.25 alors que le maximum de rappel obtenu est de 58.33.

En termes de F-mesure, les développeurs professionnels atteignent en moyenne 23.01 de F-mesure. Le minimum de F-mesure est de 5 alors que le maximum de F-mesure obtenu est de 46.15.

Le reste des statistiques descriptives comme le premier quartile, la médiane et le troisième quartile sont également reportées au niveau du tableau 3.4.

3.2 QR2 : Dans quelle mesure l'utilisation de différentes sources d'informations a-t-elle un impact sur la performance des développeurs professionnels lors de la tâche de synthèse des éléments de code ?

Dans le but de comparer les performances des développeurs professionnels en termes de précision, rappel et F-mesure selon la source d'informations utilisée, nous avons (i) généré les boîtes à moustaches relatives à chaque mesure (précision, rappel et F-mesure) et selon la source d'informations logiciel utilisée puis, (ii) nous avons utilisé le test de Wilcoxon et l'effect-size du Cliff delta entre chaque type de sources d'informations logiciel étudiées.

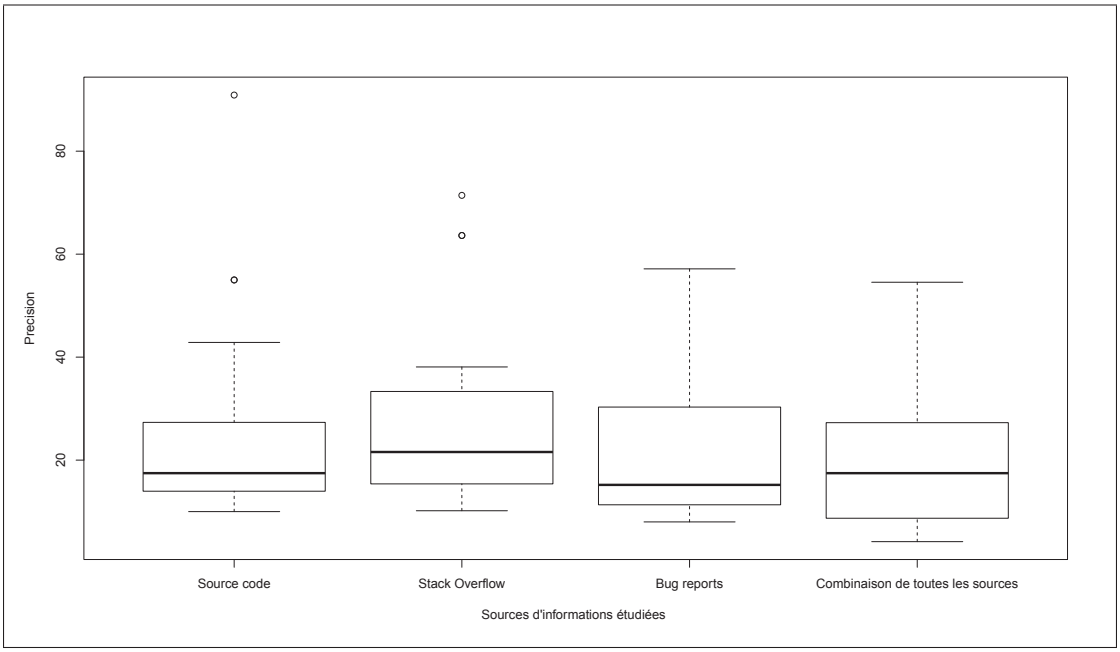


Figure 3.1 Boîtes à moustaches de la précision selon le type de sources d'informations étudiées.

La figure 3.1 illustre les boîtes à moustaches de la précision obtenue pour les quatre sources d'informations (code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations).

En se basant sur les boîtes à moustaches de la précision (figure 3.1) et les statistiques descriptives (section 3.1), nous remarquons qu'en termes de médiane, les développeurs professionnels

atteignent leurs meilleures performances lorsqu'ils utilisent comme source d'informations le Stack Overflow. Une tendance similaire a été observée pour le code source et la combinaison des sources d'informations mais avec une performance légèrement inférieure. Les performances des développeurs professionnels deviennent encore plus faibles lors de l'utilisation de rapports de bogues.

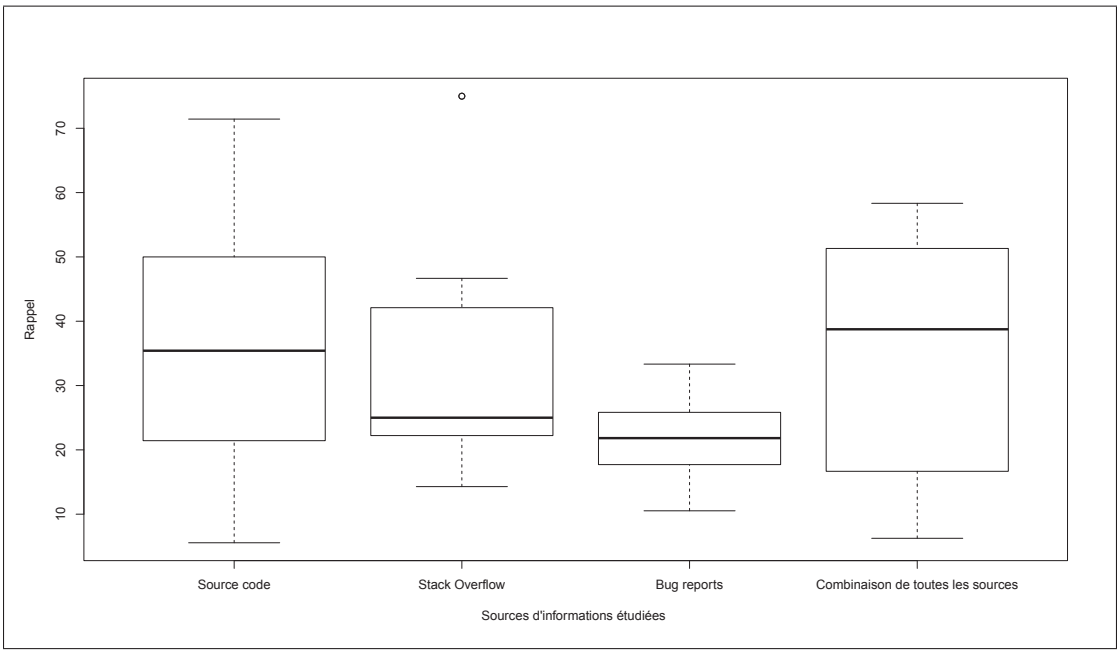


Figure 3.2 Boîtes à moustaches du rappel des différents types de sources d'informations étudiées.

La figure 3.2 illustre les boîtes à moustaches du rappel obtenu pour les quatre sources d'informations (code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations).

En se basant sur les boîtes à moustaches (figure 3.2) et les statistiques descriptives (section 3.1), nous remarquons qu'en termes de médiane, le rappel est plus élevé en utilisant la combinaison des sources d'informations puis viennent les performances du code source. Le Stack Overflow vient en troisième position alors que les plus faibles performances sont atteintes en utilisant les rapports de bogues.

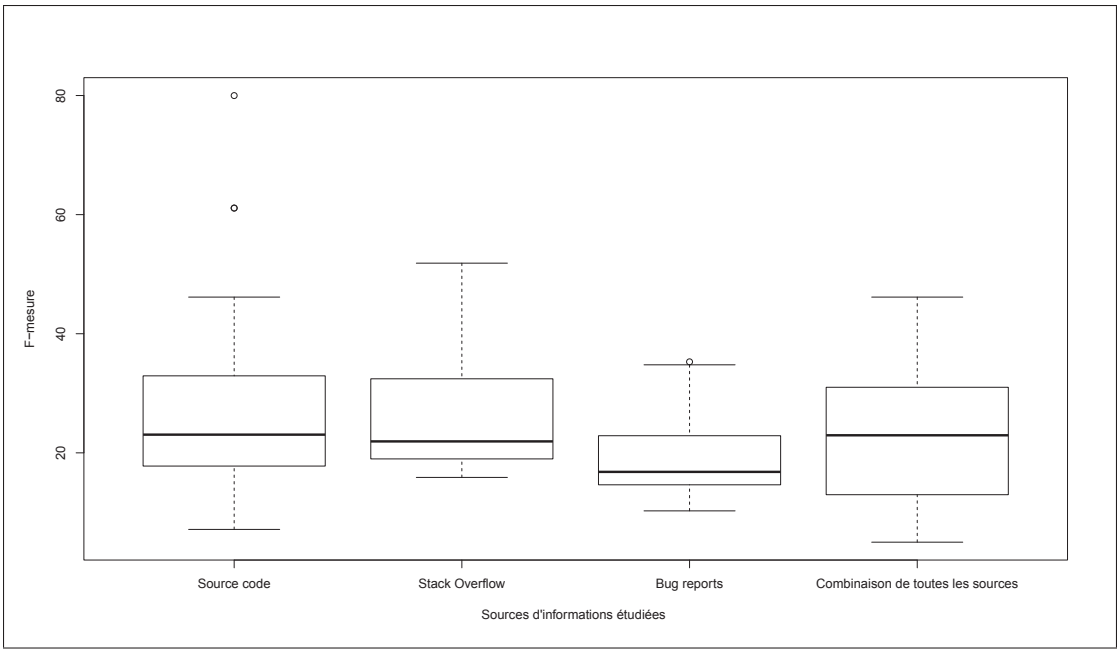


Figure 3.3 Boîtes à moustaches de la F-mesure des différents types de sources d'informations étudiées.

La figure 3.3 illustre les boîtes à moustaches de la F-mesure obtenue pour les quatre sources d'informations (code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations).

En se basant sur les boîtes à moustaches de la F-mesure (figure 3.3) et les statistiques descriptives (section 3.1), nous remarquons qu'en termes de médiane, les développeurs professionnels atteignent leurs meilleures performances lorsqu'ils utilisent le code source puis viennent les performances par la combinaison de toutes les sources. Le Stack Overflow vient en troisième position alors que les plus faibles performances sont observées pour les rapports de bogues.

Afin de déterminer s'il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation de différentes sources d'informations, nous avons utilisé le test de Wilcoxon et l'effect-size du Cliff delta entre les performances de chaque type de sources d'informations étudiées, c'est-à-dire :

- Entre le **code source** versus le Stack Overflow, versus les rapports de bogues puis versus la combinaison des sources d'informations.

- Entre le **Stack Overflow** versus le code source, versus les rapports de bogues puis versus la combinaison des trois sources d'informations.
- Entre les **rapports de bogues** versus code source, versus Stack Overflow puis versus combinaison des trois sources d'informations.
- Entre la **combinaison des trois sources d'informations** versus code source, versus le Stack Overflow puis versus les rapports de bogues.

Rappelons l'hypothèse nulle, l'hypothèse alternative et les questions spécifiques liées à la deuxième question de recherche **QR2** décrites dans la section 2.1.1 :

- $\mathbf{H_{0,QR2}}$ *: Il n'existe pas de différences significatives entre la performance des développeurs professionnels lors de l'utilisation de différents types de sources d'informations en termes de précision, rappel et F-mesure.*
- $\mathbf{H_{a,QR2}}$ *: Il existe une différence significative entre la performance des développeurs professionnels lors de l'utilisation de différents types de sources d'informations en termes de précision, rappel et F-mesure.*
- $\mathbf{QR2_1}$: Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du code source versus le Stack Overflow lors de la tâche de synthèse des éléments de code ?
- $\mathbf{QR2_2}$: Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du code source versus les rapports de bogues lors de la tâche de synthèse des éléments de code ?
- $\mathbf{QR2_3}$: Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du code source versus la combinaison des sources d'informations lors de la tâche de synthèse des éléments de code ?
- $\mathbf{QR2_4}$: Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du Stack Overflow versus rapports de bogues des sources lors de la tâche de synthèse des éléments de code ?

- **$QR2_5$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation du Stack Overflow versus la combinaison des sources d'informations des sources lors de la tâche de synthèse des éléments de code ?
- **$QR2_6$** : Existe-t-il une différence significative en termes de performance des développeurs professionnels par l'utilisation des rapports de bogues versus la combinaison des sources d'informations lors de la tâche de synthèse des éléments de code ?
- **$QR2_7$** : Existe-t-il une différence entre le temps moyen consacré par les développeurs professionnels lors des tâches de synthèse des éléments de code en utilisant les différents types de sources d'informations ?

  Les hypothèses nulles et alternatives des questions de recherche spécifiques citées ci-dessus sont détaillées dans la section 2.1.1 de ce présent document.

3.2.1 $QR2_1$ $QR2_2$ $QR2_3$ Comparaison des performances obtenues du code source versus autres sources d'informations

Tableau 3.5 Comparaison de la **précision** des synthèses obtenues par l'utilisation du **code source** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	Source d'informations 2	p-value	Cliff delta
Code source	Stack Overflow	0.464	-
Code source	Rapports de bogues	0.283	-
Code source	Combinaison des sources d'informations	0.343	-

Le tableau 3.5 rapporte les résultats de la comparaison de la précision entre le code source et les autres sources d'informations étudiées qui sont le Stack Overflow, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Les tests statistiques présentés dans le tableau 3.5 indiquent qu'il n'existe pas de différences significatives statistiquement en termes de précision lors de l'utilisation du code source versus les autres sources d'informations étudiées, c'est-à-dire, versus le Stack Overflow, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues).

Tableau 3.6 Comparaison du **rappel** des synthèses obtenues avec le **code source** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Code source	Stack Overflow	0.287	-
Code source	Rapports de bogues	**0.0021**	0.5191 (large)
Code source	Combinaison des sources d'informations	0.556	-

Le tableau 3.6 rapporte les résultats de la comparaison du rappel entre le code source et d'autres sources d'informations étudiées qui sont le Stack Overflow, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Les tests statistiques rapportés dans le tableau 3.6 indiquent qu'un résultat statistiquement significatif a été obtenu :

- Entre le code source et les rapports de bogues avec p = 0.0021 et une mesure de l'effect-size du Cliff delta égale à 0.5191 et est donc large (selon son interprétation décrite ci-dessus) en faveur du code source.

Tableau 3.7 Comparaison de la **F-mesure** des synthèses obtenues avec le **code source** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Code source	Stack Overflow	0.910	-
Code source	Rapports de bogues	**0.0115**	0.4271 (moyen)
Code source	Combinaison des sources d'informations	0.578	-

Le tableau 3.7 rapporte les résultats de la comparaison de la F-mesure entre le code source et autres sources d'informations étudiées qui sont le Stack Overflow, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Les tests statistiques rapportés dans le tableau 3.7 indiquent qu'un résultat statistiquement significatif a été obtenu pour une comparaison :

- Entre le code source et les rapports de bogues avec p = 0.0115 et une mesure de l'effect-size du Cliff delta moyenne et égale à 0.4271 en faveur du code source.

Conclusion 1 : Pour résumer les résultats obtenus à ce niveau, il n'existe pas de différences statistiquement significatives entre le code source et les autres sources d'informations étudiées en termes de précision. Par contre, il existe une différence statistiquement significative (large) en termes de rappel et une différence statistiquement moyenne en termes de F-mesure entre le code source et les rapports de bogues. Nous pouvons donc rejeter l'hypothèse nulle concernant la comparaison du code source en termes de rappel et de F-mesure seulement. Certes, les autres hypothèses nulles concernant les comparaisons du code source, le Stack Overflow ainsi que la combinaison de toutes les sources ne peuvent être rejetées.

Une seconde conclusion que nous pouvons faire des résultats non-significatifs entre le code source et le Stack Overflow du tableau 3.6 est qu'il existe d'autres sources d'informations

dans ce cas le Stack Overflow qui peuvent être utilisées pour synthétiser le but et l'usage des éléments de code.

Des sources d'informations comme les rapports de bogues sont importantes en termes de précision (résultats non-significatifs entre code source et rapports de bogues du tableau 3.5) mais ne fournissent pas suffisamment d'informations sur les éléments de code comme le code source et le Stack Overflow (résultats significatifs entre le code source et rapports de bogues dans le tableau 3.6) qui semblent être pertinents pour la tâche de synthèse.

Une autre conclusion est que l'augmentation de sources d'informations ne semble pas utile, c'est-à-dire qu'elle n'améliore pas les performances des développeurs professionnels (résultats non-significatifs rapportés dans le tableau 3.6. Une explication pour cela est que les humains ne peuvent pas gérer un large volume de données et se concentrent, en général sur un sous-ensemble d'informations quand ils sont face à diverses sources d'informations.

Nous avons calculé les performances atteintes avec l'utilisation du code source pour chaque élément de code, en termes de précision, rappel et F-mesure. Ces performances sont mentionnées dans le tableau III-1 de l'annexe III.

Pour fournir plus d'informations et un feedback additionnel concernant le code source, nous avons exploré les données fournies par les post-questionnaires qui concernent le code source. Nous avons constaté que 100% des participants étaient d'accord sur le fait que les commentaires du code source étaient utiles lors de la synthèse des éléments de code.

Nous avons constaté que 91.6% des participants étaient d'accord sur le fait que la lisibilité générale du code a aidé à la synthèse des éléments de code.

Nous avons constaté que 83.33% des participants étaient d'accord sur le fait que les lignes de code étaient utiles lors de la synthèse des éléments de code.

3.2.2 $QR2_1$ $QR2_4$ $QR2_5$ Comparaison des performances obtenues du Stack Overflow versus autres sources d'informations.

Tableau 3.8 Comparaison de la **précision** des synthèses obtenues par l'utilisation du **Stack Overflow** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Stack Overflow	Code source	0.464	-
Stack Overflow	Rapports de bogues	0.071	-
Stack Overflow	Combinaison des sources d'informations	0.137	-

Le tableau 3.8 rapporte les résultats de la comparaison de la précision entre le Stack Overflow et autres sources d'informations étudiées qui sont le code source, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

D'un point de vue statistique, les tests statistiques présentés dans le tableau 3.8 indiquent qu'il n'existe pas de différences statistiquement significatives en termes de précision lors de l'utilisation du Stack Overflow versus les autres sources d'informations étudiées, c'est-à-dire versus code source, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues).

Le tableau 3.9 rapporte les résultats de la comparaison du rappel entre le Stack Overflow et autres sources d'informations étudiées qui sont le code source, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Les tests statistiques rapportés dans le tableau 3.9 indiquent qu'un résultat statistiquement significatif a été obtenu :

Tableau 3.9 Comparaison du **rappel** des synthèses obtenues par l'utilisation du **Stack Overflow** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Stack Overflow	Code source	0.288	-
Stack Overflow	Rapports de bogues	**0.0099**	0.4340 (moyen)
Stack Overflow	Combinaison des sources d'informations	0.672	-

- Entre le Stack Overflow et les rapports de bogues avec p = 0.0099 et une mesure de l'effect-size du Cliff delta moyenne et égale à 0.4340 en faveur du Stack Overflow.

Tableau 3.10 Comparaison de la **F-mesure** des synthèses obtenues par l'utilisation du **Stack Overflow** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Stack Overflow	Code source	0.910	-
Stack Overflow	Rapports de bogues	**0.0037**	0.4896 (large)
Stack Overflow	Combinaison des sources d'informations	0.4897	-

Le tableau 3.10 rapporte les résultats de la comparaison de la F-mesure entre le Stack Overflow et autres sources d'informations étudiées qui sont le code source, les rapports de bogues et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Basés sur les tests statistiques rapportés dans le tableau 3.10, un résultat statistiquement significatif a été obtenu par la comparaison suivante :

- Entre les Stack Overflow et les rapports de bogues avec p = 0.0037 et une mesure de l'effect-size du Cliff delta large et égale à 0.4896 en faveur du Stack Overflow.

Conclusion 2 : Pour résumer les résultats obtenus à ce niveau, il n'existe pas de différences statistiquement significatives entre le Stack Overflow et les autres sources d'informations étudiées en termes de précision. Par contre, il existe une différence statistiquement significative (large) en termes de F-mesure entre le Stack Overflow et les rapports de bogues et une différence statistiquement moyenne entre le Stack Overflow et les rapports de bogues en termes de rappel.

Nous pouvons donc rejeter l'hypothèse nulle concernant la comparaison entre le Stack Overflow en termes de rappel et F-mesure seulement. Certes, les autres hypothèses nulles concernant les comparaisons du Stack Overflow, le code source ainsi que la combinaison de toutes les sources d'informations ne peuvent être rejetées.

Les résultats obtenus pour le Stack Overflow corroborent ceux obtenus pour le code source. En effet, nous apportons une évidence empirique additionnelle sur le fait que d'autres sources d'informations informelles comme le Stack Overflow peuvent être utilisées pour synthétiser le but et l'usage des éléments de code.

Des sources d'informations comme les rapports de bogues sont importantes en termes de précision (il n'existe pas de différences statistiquement significatives entre Stack Overflow et rapports de bogues comme tableau 3.8) mais ne fournissent pas suffisamment d'informations sur les éléments de code comme le code source et le Stack Overflow qui semblent être pertinents pour la tâche de synthèse (tableau 3.9.

Une autre conclusion est que l'augmentation de sources d'informations ne semble pas utile, C'est-à-dire qu'elle n'améliore pas les performances des développeurs professionnels (résultats non-significatifs rapportés dans le tableau 3.9). Une explication pour cela est que les humains ne peuvent pas probablement gérer un large volume de données et se concentrent, en général sur un sous-ensemble d'informations, quand ils sont face à diverses sources d'informations.

Nous avons calculé les performances atteintes avec l'utilisation du Stack Overflow pour chaque élément de code, en termes de précision, rappel et F-mesure. Ces performances sont mentionnées dans le tableau III-2 de l'annexe III.

Pour fournir plus d'informations et un feedback additionnel concernant l'utilisation du Stack Overflow lors de la synthèse des éléments de code, nous avons exploré les données fournies par les post-questionnaires qui concernent le Stack Overflow. Nous avons constaté que 79.16% des participants étaient d'accord sur le fait que les descriptions textuelles des publications Stack Overflow, les commentaires des utilisateurs et les exemples de code étaient utiles lors de la synthèse des éléments de code. Nous avons constaté que 66.66% des participants étaient d'accord sur le fait que les tags des questions étaient utiles lors de la synthèse des éléments de code.

3.2.3 $QR2_2$ $QR2_4$ $QR2_6$ Comparaison des performances obtenues des rapports des bogues versus autres sources d'informations

Tableau 3.11 Comparaison de la **précision** des synthèses obtenues par l'utilisation des **rapports de bogues** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	Source d'informations 2	p-value	Cliff delta
Rapports de bogues	Code source	0.283	-
Rapports de bogues	Stack Overflow	0.071	-
Rapports de bogues	Combinaison des sources d'informations	0.741	-

Le tableau 3.11 rapporte les résultats de la comparaison de la précision entre les rapports de bogues et autres sources d'informations étudiées qui sont le code source, le Stack Overflow et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

D'un point de vue statistique, les tests statistiques présentés dans le tableau 3.11 indiquent qu'il n'existe pas de différences significatives statistiquement en termes de précision lors de l'utilisation des rapports de bogues versus les autres sources d'informations étudiées, c'est-à-dire le code source, le Stack Overflow et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues).

Tableau 3.12 Comparaison du **rappel** des synthèses obtenues par l'utilisation des **rapports de bogues** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Rapports de bogues	Code source	**0.0021**	-0.5191 (large)
Rapports de bogues	Stack Overflow	**0.0099**	-0.4340 (moyen)
Rapports de bogues	Combinaison des sources d'informations	**0.0257**	-0.3767 (moyen)

Le tableau 3.12 rapporte les résultats de la comparaison du rappel entre les rapports de bogues et autres sources d'informations étudiées qui sont le code source, le Stack Overflow et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Basés sur les tests statistiques rapportés dans le tableau 3.12, les résultats statistiquement significatifs ont été obtenus par les trois comparaisons suivantes :

- Entre les rapports de bogues et le code source avec p = 0.0021 et une mesure de l'effect-size du Cliff delta égale à -0.5191 (large) en faveur du code source.
- Entre les rapports de bogues et le Stack Overflow avec p = 0.099 et une mesure de l'effect-size du Cliff delta moyenne et égale -0.4340 en faveur du Stack Overflow.
- Entre les rapports de bogues et la combinaison de toutes les sources d'informations avec une valeur de p = 0.0257 et une mesure de l'effect-size du Cliff delta égale à -0.3767 (moyenne) en faveur de la combinaison de toutes les sources d'informations.

Tableau 3.13 Comparaison de la **F-mesure** des synthèses obtenues par l'utilisation des **rapports de bogues** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Rapports de bogues	Code source	**0.0115**	-0.4271 (moyen)
Rapports de bogues	Stack Overflow	**0.0037**	-0.4896 (large)
Rapports de bogues	Combinaison des sources d'informations	0.279	-

Le tableau 3.13 rapporte les résultats de la comparaison de la F-mesure entre les rapports de bogues et autres sources d'informations étudiées qui sont le code source, le Stack Overflow et la combinaison des sources d'informations (code source, Stack Overflow et les rapports de bogues). La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Basés sur les tests statistiques rapportés dans le 3.13, les résultats statistiquement significatifs ont été obtenus pour une seule comparaison :

- Entre les rapports de bogues et le code source avec p = 0.0115 et une mesure de l'effect-size du Cliff delta moyenne avec un Cliff delta égale à - 0.4271 en faveur du code source.
- Entre les rapports de bogues et le Stack Overflow avec p = 0.0037 et une mesure de l'effect-size du Cliff delta moyenne avec un Cliff delta égale à - 0.4896 en faveur du Stack Overflow.

Nous avons calculé les performances atteintes avec l'utilisation des rapports de bogues pour chaque élément de code, en termes de précision, rappel et F-mesure. Ces performances sont mentionnées dans le tableau III-3 de l'annexe III.

Conclusion 3 : Les résultats obtenus à ce niveau nous montrent qu'il n'existe pas de différences statistiquement significatives entre les rapports de bogues et les autres sources d'informations étudiées en termes de précision.

Par contre, il existe une différence statistiquement significative (large) entre les rapports de bogues et le code source, il existe une différence statistiquement significative (moyenne) entre

les rapports de bogues et le Stack Overflow et entre les rapports de bogues et la combinaison des sources d'informations en termes de rappel. En plus, il existe une différence statistiquement significative (moyenne) entre les rapports de bogues et le code source et une différence statistiquement significative (large) entre les rapports de bogues et le Stack Overflow en termes de F-mesure. Nous pouvons donc rejeter les hypothèses nulles relatives à ces comparaisons. Par contre, nous ne pouvons pas les rejeter pour les autres instances de comparaison.

Ainsi, nous pouvons conclure que les rapports de bogues peuvent être utilisés pour synthétiser avec précision le but et l'usage des éléments de code. Par contre, cette source d'informations ne contient pas toutes les informations pertinentes aux éléments de code ce qui est reflété d'ailleurs au niveau des résultats du rappel (tableau 3.12).

Pour fournir plus d'informations et un feedback additionnel concernant l'utilisation des rapports de bogues lors de la synthèse des éléments de code, nous avons exploré les données fournies par les post-questionnaires qui concernent les rapports de bogues. Nous avons constaté que 79% des participants étaient d'accord sur le fait que les commentaires publiés par les utilisateurs des rapports de bogues étaient utiles lors de la synthèse des éléments de code. Nous avons constaté que 62.5% des participants étaient d'accord sur le fait que les exemples de code des rapports de bogues étaient utiles lors de la synthèse des éléments de code.

3.2.4 $QR2_3$ $QR2_5$ $QR2_6$ Comparaison des performances obtenues de la combinaison des sources d'informations versus le code source, les rapports de bogues et le Stack Overflow

Le tableau 3.14 rapporte les résultats de la comparaison de la précision entre la combinaison des sources d'informations et le code source, les rapports de bogues et le Stack Overflow. La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

D'un point de vue statistique, les tests statistiques présentés dans le tableau 3.14 indiquent qu'il n'existe pas de différences significatives statistiquement en termes de précision lors de l'utili-

Tableau 3.14 Comparaison de la **précision** obtenue de la **combinaison des sources d'informations** versus le code source, les rapports de bogues et le Stack Overflow : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Combinaison des sources d'informations	Code source	0.343	-
Combinaison des sources d'informations	Stack Overflow	0.137	-
Combinaison des sources d'informations	Rapports de bogues	0.741	-

sation de la combinaison des sources d'informations versus les autres sources d'informations étudiées, c'est-à-dire le code source, le Stack Overflow et les rapports de bogues.

Tableau 3.15 Comparaison du **rappel** des synthèses obtenues de la **combinaison des sources d'informations** versus autres sources d'informations : Résultats du test de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Combinaison des sources d'informations	Code source	0.556	-
Combinaison des sources d'informations	Stack Overflow	0.672	-
Combinaison des sources d'informations	Rapports de bogues	**0.0257**	0.3767 (moyen)

Le tableau 3.15 rapporte les résultats de la comparaison du rappel entre la combinaison des sources d'informations et le code source, les rapports de bogues et le Stack Overflow. La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Basés sur les tests statistiques rapportés dans le tableau 3.15, le résultat statistiquement significatif obtenu par une comparaison est le suivant :

- Entre la combinaison de toutes les sources d'informations et les rapports de bogues avec une valeur de p = 0.0257 et une mesure de l'effect-size du Cliff delta moyenne et égale à 0.3767 en faveur de la combinaison des sources d'informations.

Tableau 3.16 Comparaison de la **F-mesure** des synthèses obtenues de la **combinaison des sources d'informations** versus autres sources d'informations : Résultats du test apparié de Wilcoxon et de l'effect-size du Cliff delta.

Source d'informations 1	**Source d'informations 2**	**p-value**	**Cliff delta**
Combinaison des sources d'informations	Code source	0.578	-
Combinaison des sources d'informations	Stack Overflow	0.4897	-
Combinaison des sources d'informations	Rapports de bogues	0.279	-

Le tableau 3.16 rapporte les résultats de la comparaison de la F-mesure entre la combinaison des sources d'informations et le code source, les rapports de bogues et le Stack Overflow. La comparaison est réalisée par l'utilisation du test de Wilcoxon et de l'effect-size du Cliff delta.

Les résultats ont démontré qu'il n'existe pas de différences statistiquement significatives en termes de performances des développeurs professionnels lors de la synthèse des éléments de code entre l'utilisation des différents types de sources d'informations (Code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations), puisque les développeurs professionnels ont obtenu presque les mêmes performances avec l'utilisation des quatre sources d'informations (code source, Stack Overflow, rapports de bogues et leur combinaison).

Conclusion 4 : Les résultats obtenus à ce niveau apportent une évidence empirique sur le fait que l'augmentation d'informations lors de la tâche de synthèse des éléments de code n'améliore pas la performance des développeurs professionnels. Malgré la différence statistiquement significative obtenue par la comparaison de la combinaison des sources d'informations avec

les rapports de bogues en faveur de la combinaison des sources d'informations, cette différence n'est pas reflétée au niveau de la F-mesure qui est l'agrégation de la précision et du rappel. Nous pensons que d'autres résultats peuvent être obtenus lors des tâches de synthèse par des techniques et outils automatiques vu qu'ils peuvent gérer automatiquement une variété de données contrairement aux humains qui se surchargent de grandes quantités d'informations (Guerrouj, 2013).

Nous avons calculé les performances atteintes avec l'utilisation de la combinaison des sources d'informations pour chaque élément de code, en termes de précision, rappel et F-mesure. Ces performances sont mentionnées dans le tableau III-4 de l'annexe III.

3.2.5 QR2$_7$ Existe-t-il une différence entre le temps moyen consacré par les développeurs professionnels lors des tâches de synthèse des éléments de code en utilisant les différents types de sources d'informations ?

Grâce aux enregistrements effectués avec l'outil *Camtasia Studio*, nous avons pu calculer le temps moyen que les développeurs professionnels ont passé lors de la réalisation des tâches de synthèse. Afin de synthétiser un élément de code les développeurs ont consacré en moyenne :

- 9 minutes 30 secondes en utilisant le code source.
- 10 minutes 58 secondes en utilisant Stack Overflow.
- 10 minutes 12 secondes en exploitant les rapports de bogues.
- 14 minutes 33 secondes en combinant toutes les sources d'informations.

Il est clair que les développeurs professionnels passent plus de temps à synthétiser l'élément de code en utilisant la combinaison des sources d'informations, alors qu'ils passent moins de temps à synthétiser les éléments de code en utilisant le code source, Stack Overflow ou les rapports de bogues avec de légères différences (quantifiées en secondes).

Conclusion 5 : Le temps moyen consacré par les développeurs professionnels lors des tâches de synthèse des éléments de code en utilisant le code source, le Stack Overflow ou les rapports de bogues est similaire.

Dans ce qui suit, nous présentons notre conclusion générale :

Conclusion générale

La tâche de synthèse des éléments de code peut tirer bénéfice des sources d'informations informelles autres que les sources formelles classiques telles que le code source qui a été largement exploité par les travaux de recherches antérieures dans ce domaine, par exemple, les travaux menés par Moreno & Aponte (2012) et Moreno *et al.* (2014). Ces sources d'informations informelles telles que le Stack Overflow, les rapports de bogues ou la combinaison des sources d'informations aident à générer des synthèses du but et de l'usage des éléments de code qui sont context-aware et peuvent ainsi augmenter la documentation officielle qui d'après les travaux ultérieurs (Treude & Robillard, 2016) manque de contexte et de complétude.

L'augmentation de contexte pour les humains ne semble pas être utile pour les développeurs professionnels quand ils réalisent la tâche de synthèse manuellement. Certes, nous faisons l'hypothèse que les outils de synthèses automatiques peuvent gérer une variété d'informations (contrairement aux humains) telle que mentionnée dans les travaux menés par Guerrouj *et al.* (2015) sur l'effet du contexte.

Le temps moyen consacré par les développeurs professionnels lors des tâches de synthèse des éléments de code en combinant les sources d'informations est clairement plus élevé que lors de l'utilisation des sources d'information (code source, Stack Overflow et rapports de bogues).

Les chercheurs et les praticiens qui sont intéressés à créer des techniques et outils de synthèse des éléments de code peuvent ainsi intégrer les résultats de ce travail au niveau de leurs approches.

3.3 Obstacles à la validité

Dans cette section, nous présentons les obstacles à la validité de notre approche, par la présentation de la validité externe, la validité interne, la validité de construction et la validité de conclusion.

3.3.1 Validité externe

Les menaces à *la validité externe* concernent la possibilité de généraliser nos résultats. Pour rendre nos résultats aussi généralisables que possible, nous avons pris en considération une multitude de prédispositions afin de contrer les menaces de validités externes telles que le choix des projets à partir desquels des éléments de code ont été échantillonnés et les particularités des développeurs professionnels, etc. Cela est discuté dans ce qui suit.

En ce qui concerne l'échantillonnage des éléments de code, nous avons sélectionné aléatoirement notre échantillon d'éléments de code à partir d'un ensemble de quatre projets open-source : Eclipse, JMeter, Tomcat et NetBeans. Afin de (i) diversifier les classes et les méthodes issues de ces derniers et de (ii) généraliser les résultats obtenus, nous considérons le choix de quatre projets open-source comme étant un nombre raisonnable de projets à étudier et que cela peut être représentatif des éléments de code existants.

Le choix des classes et des méthodes est considéré comme étant une menace à la validité externe, car les résultats obtenus à partir de ces huit éléments de code peuvent ne pas être généralisés. Pour contrer cette menace, nous avons sélectionné aléatoirement ces éléments de code à partir de quatre projets open-source (Eclipse, NetBeanse, Tomcat et JMeter).

Le nombre d'éléments de code limité à huit peut aussi être considéré comme étant une menace à la validité externe. Ce nombre a été défini en relation avec (i) le nombre de tâches à présenter aux participants qui devait être raisonnable afin de ne pas générer une fatigue et avec (ii) le nombre de séquences qui est fixé à 24 séquences.

Le nombre de participants peut s'avérer insuffisant pour généraliser les résultats obtenus des synthèses qu'ils ont générés. Cette menace à la validité externe est particulièrement difficile à contrer, car il est compliqué de recruter un nombre important de participants, de par (i) la difficulté de trouver des volontaires à l'expérimentation contrôlée, (ii) l'expérimentation contrôlée se déroule dans une salle précise ce qui oblige les participants à se déplacer (contrairement aux expérimentations en ligne qui attirent plus de participants de par les multiples facilités qu'elles offrent), (iii) de s'adapter aux disponibilités de chaque participant. Il est obligatoire de fixer des horaires bloqués selon les disponibilités des participants avec parfois une heure de décalage, ce qui peut s'avérer important avec un nombre de participants élevé et (iiii) les volontaires ont un profil de développeurs professionnels qui restreint le nombre de participants. Afin de limiter cette menace, nous avons opté au recrutement des participants d'un certain niveau de programmation avec comme contrainte de faire participer des développeurs professionnels travaillant dans des entreprises canadiennes. Ce qui assure l'expertise des participants. Les participants qui ont réalisé l'expérimentation contrôlée appartiennent à une population de développeurs professionnels canadiens. Les participants sont issus de 16 entreprises et institutions canadiennes, dont CGI, Deloitte, la société Ciena, B-CITI, Fujitsu Conseil Canada, Gestisoft, 360Medlink, Alithya, Banque Nationale du Canada, Rideau Recognition Solutions, GIRO, Ooda Technologies, SecureOps, CAE, PSP Investments et ministère des Services partagés Canada, branche : Superinformatique. Les participants sont familiers avec les tâches de maintenance et d'évolution du logiciel.

Enfin, le choix du langage de programmation peut atteindre la généralisation de nos résultats, car nous avons choisi de traiter des éléments de code en langage de programmation *Java*, ce qui implique (i) un manque d'expertise possible, car les développeurs professionnels se sont auto-évalués. Ce qui ne représente pas forcément une évaluation objective, de plus, trois participants se considèrent comme étant fiables en programmation *Java*, (ii) les participants peuvent maîtriser d'autres langages de programmation ce qui peut biaiser leurs compréhensions du code source et peut interférer dans la qualité des synthèses fournies par les participants et (iii) le langage de programmation *Java* ne représente qu'un seul langage de programmation

parmi tant d'autres. Par contre, nous considérons le langage de programmation *Java* comme étant le langage de programmation le plus utilisé et qu'un développeur professionnel possède une certaine facilité à s'adapter d'un langage de programmation à un autre en ce qui concerne la compréhension. En plus, si nous choisissons de traiter d'autres langages de programmation, cela impliquerait le recrutement de plus de participants ainsi qu'un large éventail d'éléments de code. Dans ce cas de figure, le niveau de difficulté de la réalisation de l'expérimentation contrôlée sera plus important.

3.3.2 Validité interne

Les menaces à la *validité interne* concernent tous les facteurs de confusion pouvant influencer nos résultats. Un des cas de menace de validité interne est celui lié à l'apprentissage et à l'effet de fatigue des participants. Cette menace est contrôlée dans notre expérimentation en utilisant différents traitements qui sont :

Donner aux participants peu de tâches à effectuer : En ne présentant que quatre éléments de code à synthétiser, soit deux classes et deux méthodes pour chaque participant où ils ont consacré un temps moyen de 45 minutes et 13 secondes, ce qui représente un temps raisonnable de concentration.

Générer aléatoirement l'ordre de présentation des tâches pour chaque séquence : Nous avons généré aléatoirement l'ordre de présentation des tâches afin de permettre une diversification de la présentation des questionnaires et de ne pas permettre aux participants de se concentrer qu'à une seule tâche.

Diversifier les sources d'informations : À Chaque participant est demandé d'utiliser pour chaque tâche une source d'informations différente. C'est-à-dire qu'il ne peut y avoir deux tâches identiques telles que l'utilisation du code source afin de synthétiser deux classes différentes.

Diversifier les classes et méthodes : Proposer aux participants de synthétiser deux classes et deux méthodes différentes.

Diversifier les projets : En proposant à chaque participant différents projets. Avec l'obligation qu'un participant ne peut avoir à synthétiser quatre éléments de code issus d'un même projet.

Proposer des classes et méthodes difficiles : Nous avons jugé que les classes et méthodes utilisées dans l'expérimentation contrôlée sont relativement compliquées et peu utilisées. Cette constatation s'est confirmée avec les participants où ils ont précisé qu'ils n'ont jamais utilisé ces classes et méthodes dans le cadre de leur travail ou projet et qu'ils n'avaient aucune idée de leur utilité avant d'avoir consulté la documentation de l'expérimentation contrôlée.

Faciliter l'expérimentation : Les participants ont trouvé l'expérimentation contrôlée modérément difficile (3 sur 5 sur l'échelle de Likert). Ce qui est acceptable pour une expérimentation contrôlée de cette envergure.

Autre cas de menace à la validité interne, le manque d'informations liées à quelques éléments de code. Lors de la consultation du code source, des rapports de bogues ou des publications de Stack Overflow, il est possible que les participants trouvent peu de ressources d'informations telles que les trois exemples suivants :

- Le cas de la recherche dans le code source de la **méthode2** : *org.apache.jmeter.Jmeter.convertSubTree* où il n'existe que 14 lignes de code et 18 lignes de commentaires.
- Dans le cas de la **méthode4** : *org.netbeans.api.progress.ProgressUtils.DifficilerunOffEventDispatchThread*, ou la recherche des participants n'avaient qu'une seule publication Stack Overflow à consulter.
- Dans le contexte des rapports de bogues, l'exemple de la **méthode3** : *org.apache.catalina.realm.JDBCRealm.getRoles* ou la recherche des participants ne donnait qu'un seul rapport de bogues à consulter.

Cette menace peut avoir une incidence sur la qualité de leurs synthèses. Nous avons conservé ces éléments de code, car ils reflètent des situations réalistes.

La menace à la validité interne peut être liée à la difficulté des méthodes et des classes. Cette menace est inévitable, car si nous choisissons des éléments de code moyennement faciles à comprendre, cela ne nécessitera pas une documentation importante. Il peut être facile à synthétiser sans consultation de la documentation. Le tableau 2.3 indique que les méthodes et classes synthétisées dans cette étude sont généralement difficiles.

3.3.3 Validité de construction

Les menaces de la validité de construction concernent la relation entre la théorie et l'observation. Dans notre étude, cette menace est principalement due à des erreurs possibles dans l'oracle, que nous ne pouvons exclure à priori. Pour limiter une telle menace, l'oracle a été produit en utilisant une approche consensuelle, c'est-à-dire que trois annotateurs ont produit des oracles indépendants, puis ils ont construit un seul oracle issu des trois premiers oracles. Pour tous les cas où il y a eu un désaccord concernant les synthèses obtenues, une discussion a eu lieu entre deux annotateurs puis un consensus a été atteint. Les annotateurs sont composés de l'auteure de ce mémoire ainsi que les deux professeures Olga Baysal (Université de Carleton, Ottawa, Canada) et Bonita Sharif (Université de Youngstown State, Ohio, États-Unis).

Autre point non considéré, le niveau en langue anglaise des 24 développeurs professionnels. Il est évident que par leur expertise et leurs emplois actuels, ils sont dans l'obligation d'avoir un niveau en langue anglaise acceptable.

3.3.4 Validité de conclusion

Les menaces à la validité de conclusion concernent des problèmes qui affectent la capacité à tirer les bonnes conclusions sur les relations entre le traitement et le résultat de l'expérimentation contrôlée. Nous avons utilisé des tests non paramétriques, c'est-à-dire des tests de Wilcoxon et de permutation, qui ne font aucune hypothèse sur les distributions sous-jacentes

de l'ensemble de données. En outre, chaque fois que plusieurs tests Wilcoxon sont effectués, nous avons ajusté les p-values en utilisant la correction de Holm. Enfin, nous avons utilisé la mesure de l'effect-size non paramétrique, c'est-à-dire le Cliff delta, qui est définie comme la probabilité qu'un membre d'un échantillon choisi de façon aléatoire qui a une réponse plus élevée qu'un membre d'un second échantillon, moins la probabilité inverse.

3.4 Limites de l'approche

Malgré les résultats décrits dans la section ci-dessus, plusieurs limites ont été constatées :

Nombre de participants restreint : Même si la participation de 24 développeurs professionnels est considérée comme étant satisfaisante vu la difficulté de recruter des participants, il serait préférable d'étendre l'expérimentation sur une population plus importante de développeurs professionnels afin de généraliser nos résultats avec la participation d'une plus grande variété de qualifications et d'expertises.

Nombre d'éléments de code restreint : Dans notre étude, nous avons sélectionné huit éléments de code issus de quatre projets *Java* open-source (Eclipse, Netbeanse, Tomcat et JMeter). Il serait préférable de considérer plus d'éléments de code afin de généraliser les résultats obtenus.

Manque d'informations relatives à la documentation formelle et informelle : Les méthodes et classes choisies aléatoirement ont montré que la documentation formelle ou informelle peut être faible en informations, ce qui reflète la réalité de la documentation existante. Malgré cela, il n'existe pas de différences significatives entre la performance des participants par l'utilisation des différentes sources d'informations lors de la synthèse des éléments de code.

Choix du langage de programmation : Les éléments de code choisis pour la tâche de synthèse sont issus de projets open-source du langage de programmation *Java*. Il serait souhaitable de recruter des participants ayant un niveau très élevé en programmation *Java* ou de proposer des éléments de code issus de divers langages de programmation tels que Python, C++, PHP, etc.

CHAPITRE 4

CONCLUSION

L'expérimentation contrôlée présentée dans ce mémoire analyse les performances de développeurs professionnels à synthétiser des éléments de code, en utilisant une variété de sources d'informations (code source, Stack Overflow, rapports de bogues et la combinaison des sources d'informations), dans le but de montrer s'il existe des différences lors de l'utilisation de ces types de sources d'informations et quelle est la source d'informations la plus pertinente. Les résultats de ce travail peuvent être exploités par des chercheurs et praticiens qui sont intéressés par la construction d'outils de synthèse automatiques de code en langage naturel. Pour ce faire, nous avons mené une expérimentation contrôlée sur 24 développeurs professionnels issus de compagnies et institutions canadiennes. Cette expérimentation contrôlée consistait principalement à inviter les participants à synthétiser des éléments de code appartenant à quatre projets open-source *Java* (Eclipse, JMeter, Tomcat, Netbeans) par l'utilisation des sources d'informations, dont le code source, les rapports de bogues, le Stack Overflow ou la combinaison de ces sources d'informations (code source, les rapports de bogues et le Stack Overflow).

Notre travail novateur a démontré que les sources d'informations informelles, en particulier le Stack Overflow et les rapports de bogues, peuvent être utilisées pour synthétiser les éléments de code. Alors que les résultats montrent que les développeurs professionnels performent mieux la tâche de synthèse d'éléments de code en utilisant les sources d'informations logiciel de code source et de Stack Overflow, l'analyse statistique n'a pas montré une différence statistique quant à l'exactitude des synthèses fournies par les développeurs lors de l'utilisation du code source, du Stack Overflow, des rapports de bogues ou de la combinaison des sources d'informations logiciel. Donc, nous pouvons nous baser sur d'autres sources d'informations logiciel que sur le code source, telles que le Stack Overflow, les rapports de bogues et leur combinaison lors de la synthèse des éléments de code. Ce qui implique que les futurs outils automatiques peuvent prendre en considération ces sources d'informations afin, de réaliser la tâche de synthèse.

Par contre, les rapports de bogues ne fournissent pas suffisamment d'informations sur les éléments de code comme le code source et le Stack Overflow qui semblent être pertinents pour la tâche de synthèse.

Étonnamment, l'augmentation des informations pour les développeurs professionnels n'a pas amélioré leur performance en synthèse des éléments de code. Ils synthétisent avec relativement la même performance de par la combinaison des sources d'informations logiciel que par l'utilisation d'une seule source d'informations (code source, Stack Overflow ou rapports de bogue). Ceci peut s'expliquer par le fait qu'une grande quantité d'informations peut être difficile à assimiler par les humains et ils choisissent un sous-ensemble d'informations pour faire la tâche de synthèse.

Intuitivement, les développeurs professionnels passent plus de temps à synthétiser les éléments de code en utilisant une combinaison des sources d'informations logiciel (code source, Stack Overflow et les rapports de bogues), alors qu'ils ont consacré environ le même temps en synthétisant les éléments de code par l'utilisation de code source, des rapports de bogues ou du Stack Overflow.

Comme travaux futurs, il serait pertinent de créer les outils de synthèse automatique du code et de documentation, capables de tirer parti des informations nécessaires pour obtenir des synthèses précises qui aideront les développeurs dans leurs tâches de synthèse. Ces outils seront d'abord développés, puis intégrés dans des outils de développement d'environnement pour aider les développeurs dans des contextes pratiques. De plus, il serait intéressant d'étendre ce travail à une expérimentation contrôlée à grande échelle pour généraliser nos résultats et examiner si les caractéristiques des développeurs (par exemple, l'expérience, la programmation *Java*, etc.) affectent l'exactitude des synthèses fournies. Entre autres, il serait intéressant de mener d'autres études pour comprendre comment les développeurs synthétisent les éléments de code en utilisant ces sources d'informations et quel type d'informations est plus pertinent en utilisant des outils tels que l'Eye Tracking qui a la possibilité de suivre avec exactitude la recherche des participants.

Ce travail a fait l'objet d'un article de conférence qui a été soumis (23 Octobre 2017) à la conférence ACM/IEEE ICSE 2018 (40th International Conference On Software Engineering). Il a été réalisé avec la collaboration de la Professeure Olga Baysal (Université de Carleton, Ottawa, Canada) et de la Professeure Bonita Sharif (Université de Youngstown State, Ohio, États-Unis).

ANNEXE I

LIENS VERS LES 24 QUESTIONNAIRES

Le tableau I-1 présente les liens qui mènent vers chaque questionnaire expérimental.

Tableau-A I-1 Liens vers les 24 questionnaires expérimentaux.

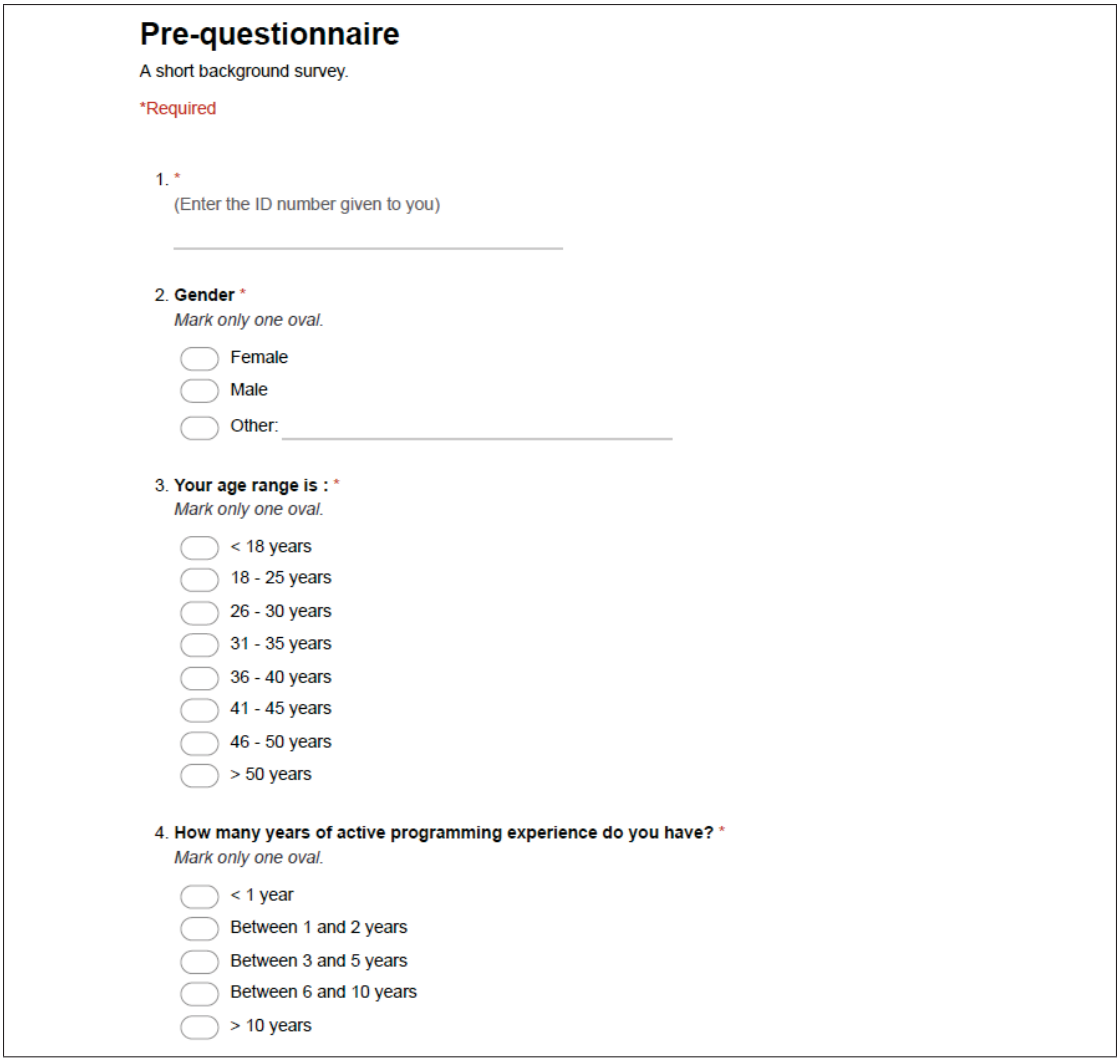
# Séquence	Liens vers les questionnaires principaux
1	https://goo.gl/forms/4Ek3qOs2ZngMZJO02
2	https://goo.gl/forms/AiXBLVTRI28hjO4r2
3	https://goo.gl/forms/x5rd63wW9VhY0j0p2
4	https://goo.gl/forms/Icgt1boZTJuA7x5A3
5	https://goo.gl/forms/UTGvAbF1jWzoJNQE2
6	https://goo.gl/forms/IRUJaXCHHjW2M67s2
7	https://goo.gl/forms/1hKIznRt8Nob8iBp1
8	https://goo.gl/forms/QzsgyFJ0v9DF490C2
9	https://goo.gl/forms/aSlRRyDjPzWhbJXW2
10	https://goo.gl/forms/PnLoad6nVhS8jsao1
11	https://goo.gl/forms/gmkl0fN4PNup6xBS2
12	https://goo.gl/forms/qTi7tQEpmvcwcTgr1
13	https://goo.gl/forms/DXHe7SxmwlkfiN9D2
14	https://goo.gl/forms/Ld5oswI0jzcYjxn02
15	https://goo.gl/forms/qNsqiZ74EMoSi8Zj2
16	https://goo.gl/forms/Bt4QEUFNt2eQYHKv2
17	https://goo.gl/forms/Mupxx4kbiZp0YENp2
18	https://goo.gl/forms/GZc8cbmrUZnX6WTu1
19	https://goo.gl/forms/WX8ddCissTjiaIfp2
20	https://goo.gl/forms/fjEzCX33U3MBTe9d2
21	https://goo.gl/forms/b9hpIXNpTaIfgB9o1
22	https://goo.gl/forms/fn8cWGAMFIWgfE1l1
23	https://goo.gl/forms/2IXQLiDjqV0dmiqh2
24	https://goo.gl/forms/Mu5lBqPoT3WLXawd2

ANNEXE II

QUESTIONNAIRES PRÉSENTÉS AUX PARTICIPANTS

1. Pré-questionnaire

Les figures II-1, II-2 et II-3 illustrent les questions posées aux développeurs professionnels qui constituent le pré-questionnaire.



Pre-questionnaire

A short background survey.

*Required

1. *
(Enter the ID number given to you)

2. **Gender** *
Mark only one oval.
- Female
- Male
- Other:

3. **Your age range is :** *
Mark only one oval.
- < 18 years
- 18 - 25 years
- 26 - 30 years
- 31 - 35 years
- 36 - 40 years
- 41 - 45 years
- 46 - 50 years
- > 50 years

4. **How many years of active programming experience do you have?** *
Mark only one oval.
- < 1 year
- Between 1 and 2 years
- Between 3 and 5 years
- Between 6 and 10 years
- > 10 years

Figure-A II-1 Partie 1 : Pré-questionnaire.

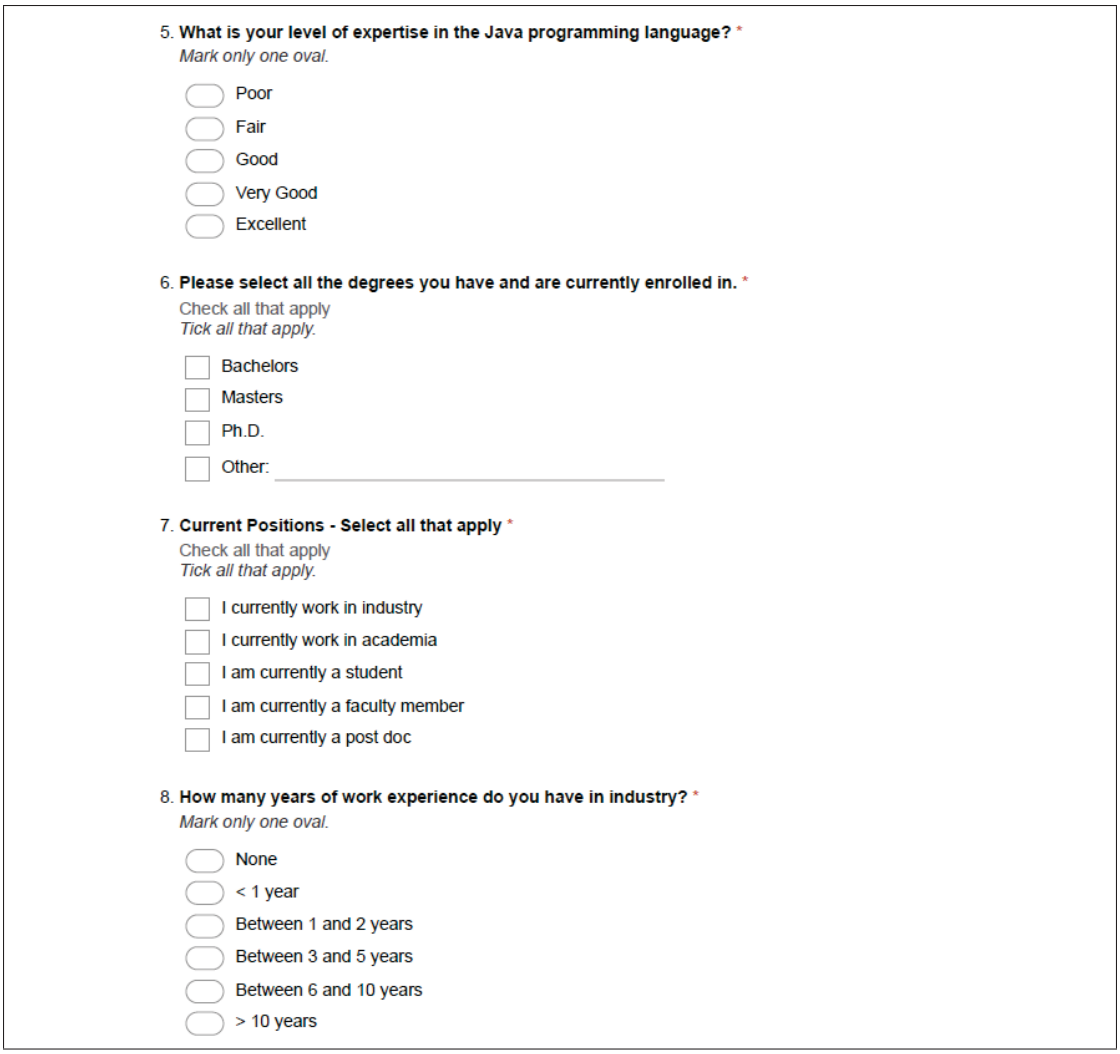


Figure-A II-2 Partie 2 : Pré-questionnaire.

9. **Do you use Stack Overflow to find solutions to your coding problems?** *
Mark only one oval.

- ◯ Yes
- ◯ No

10. **Do you use/read bug reports to find solutions to issues while coding?** *
Mark only one oval.

- ◯ Yes
- ◯ No

11. **Have you contributed (code and/or documentation) to an open source project?** *
Mark only one oval.

- ◯ Yes
- ◯ No

12. **Which of the following IDEs are you familiar with? (By familiar we mean you are able to work in fairly well).** *
Check all that apply
Tick all that apply.

- ☐ Eclipse
- ☐ Visual Studio
- ☐ Netbeans
- ☐ IntelliJ
- ☐ Other: _____

Thank you for your participation!

Figure-A II-3 Partie 3 : Pré-questionnaire.

2. Questionnaire expérimental

Les figures II-4, II-5, II-6 et II-7 illustrent les tâches expérimentales qui constituent le questionnaire expérimental.

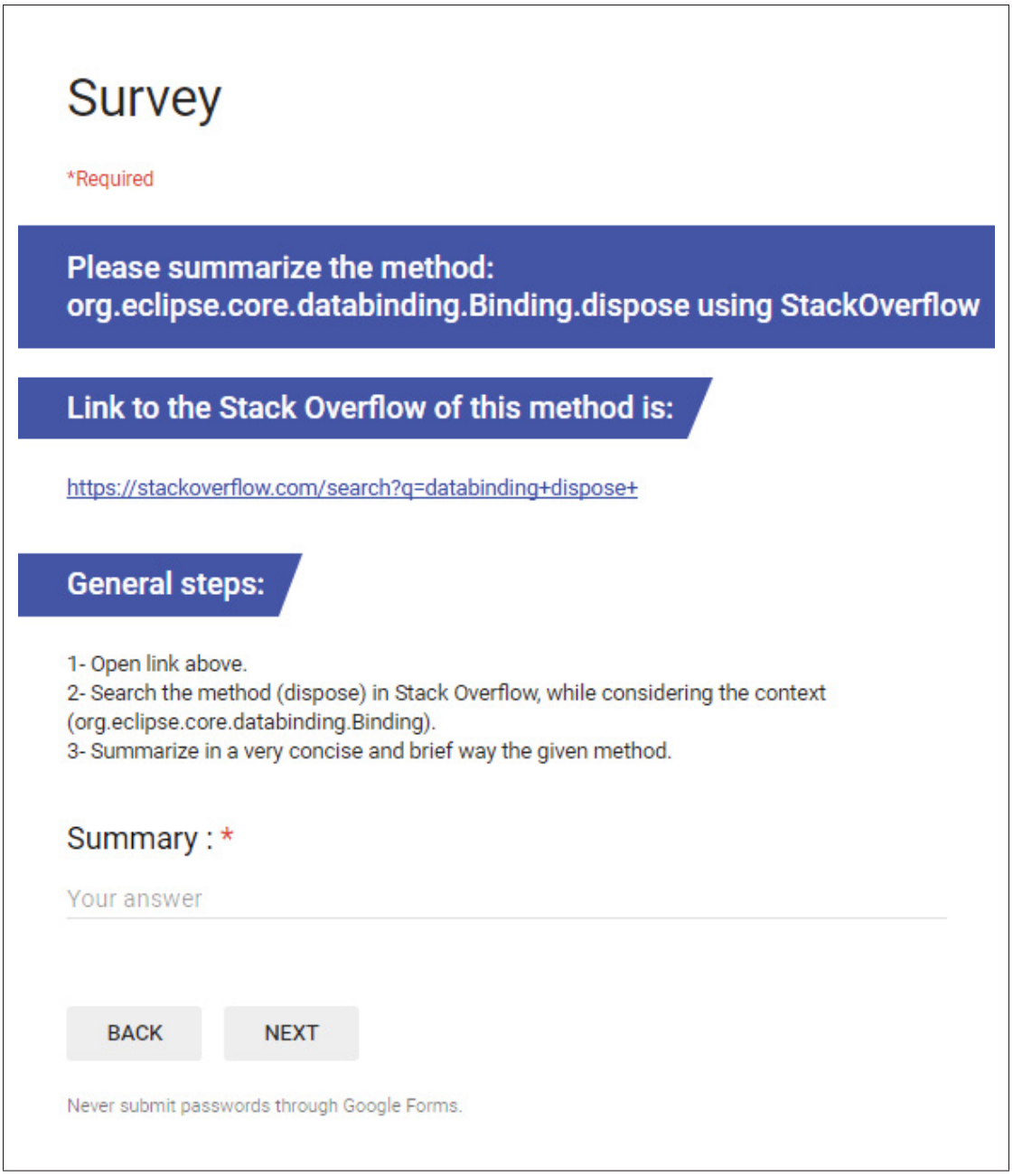


Figure-A II-4 Section du questionnaire expérimental représentant la première tâche.

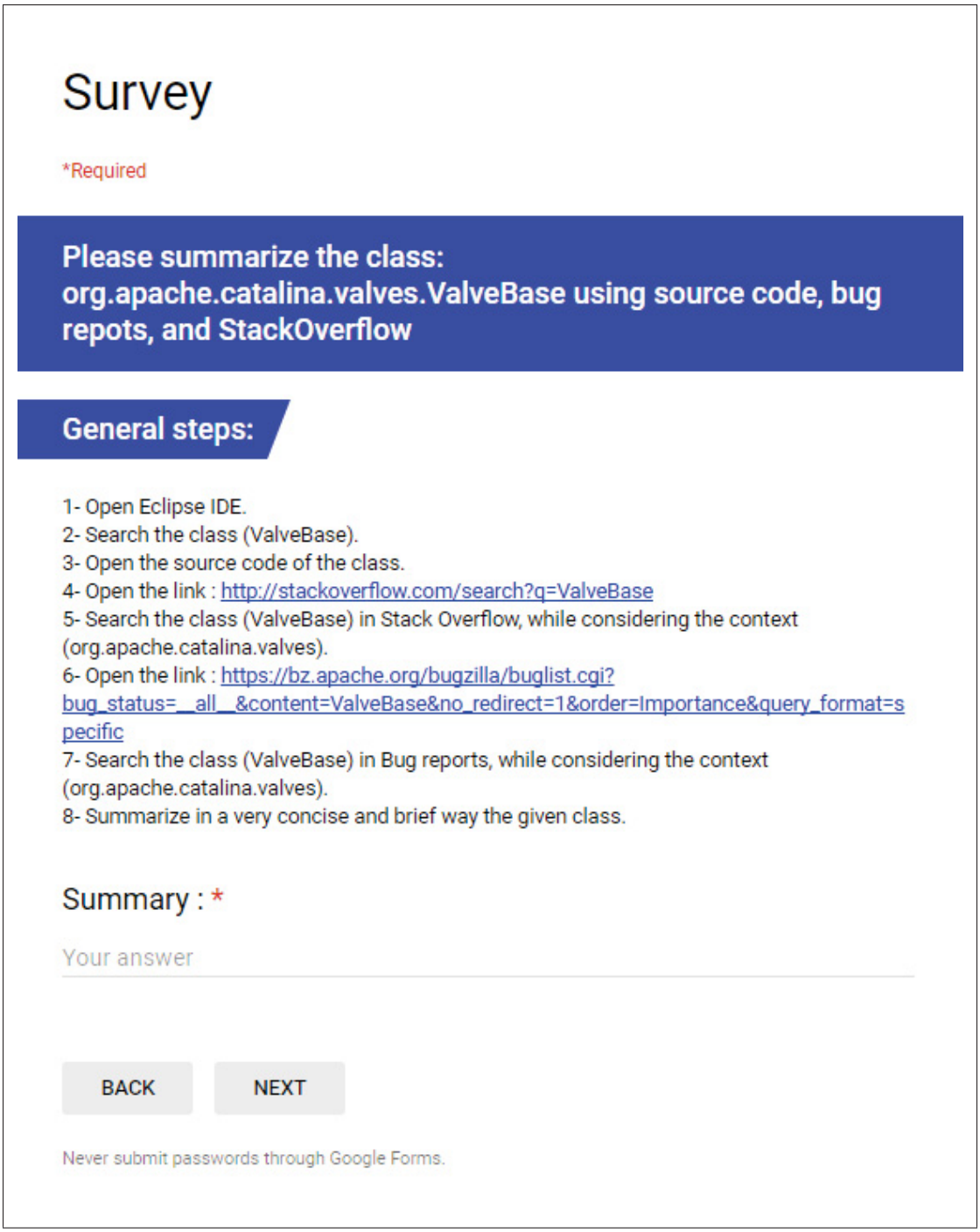


Figure-A II-5 Section du questionnaire expérimental représentant la deuxième tâche.

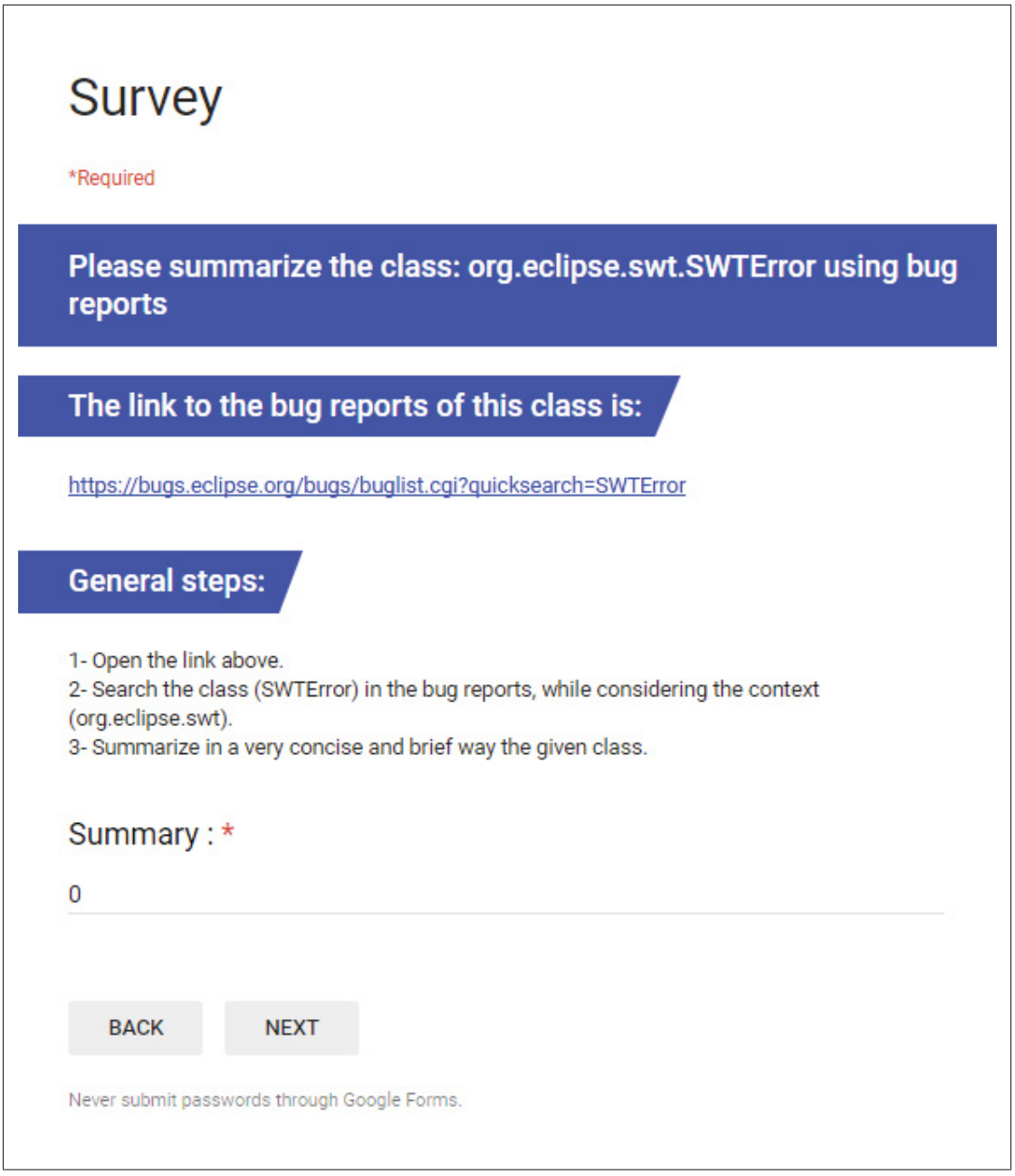


Figure-A II-6 Section du questionnaire expérimental représentant la troisième tâche.

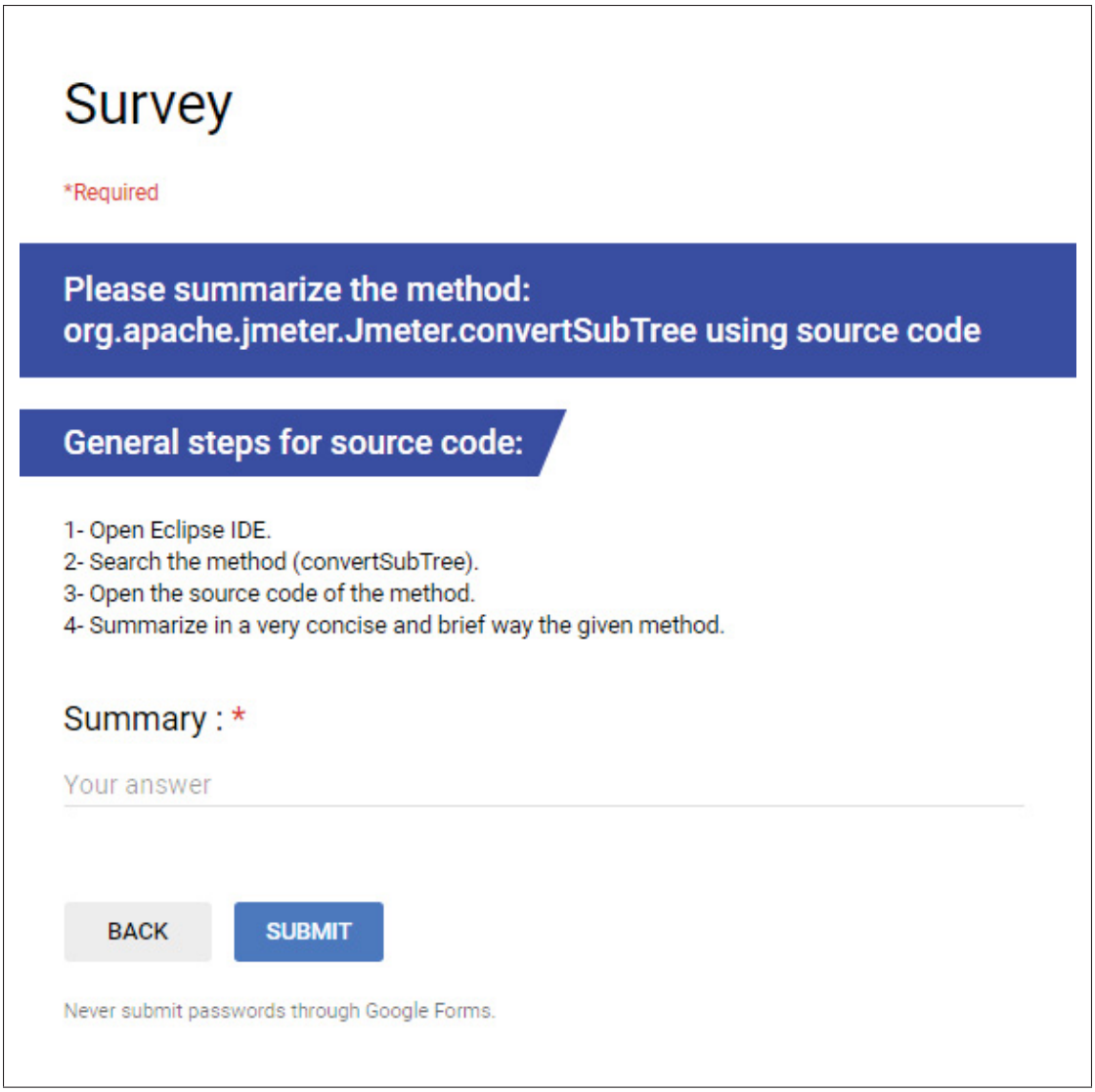
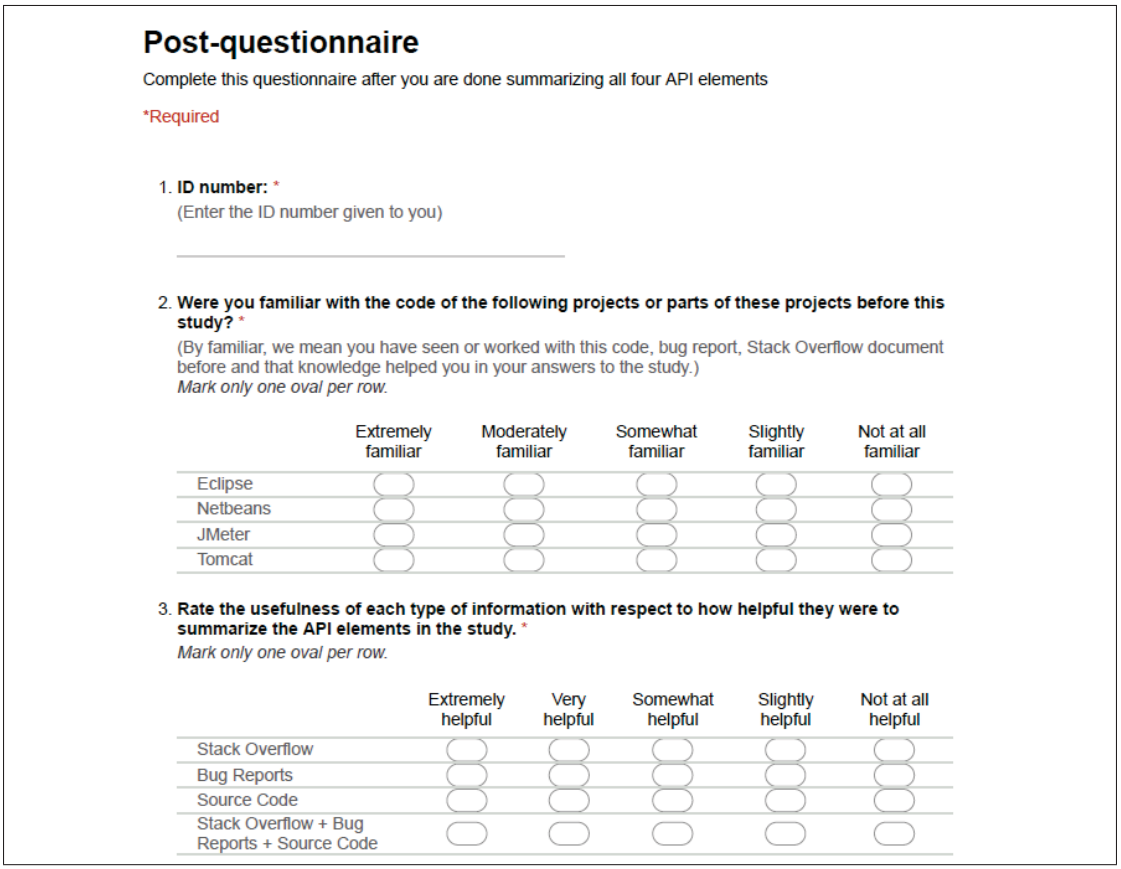


Figure-A II-7 Section du questionnaire expérimental représentant la quatrième tâche.

3. Post-questionnaire

Les figures II-8, II-9 et II-10 illustrent les questions posées aux développeurs professionnels qui constituent le post-questionnaire de l'expérimentation contrôlée.



Post-questionnaire

Complete this questionnaire after you are done summarizing all four API elements

*Required

1. **ID number:** *
   (Enter the ID number given to you)

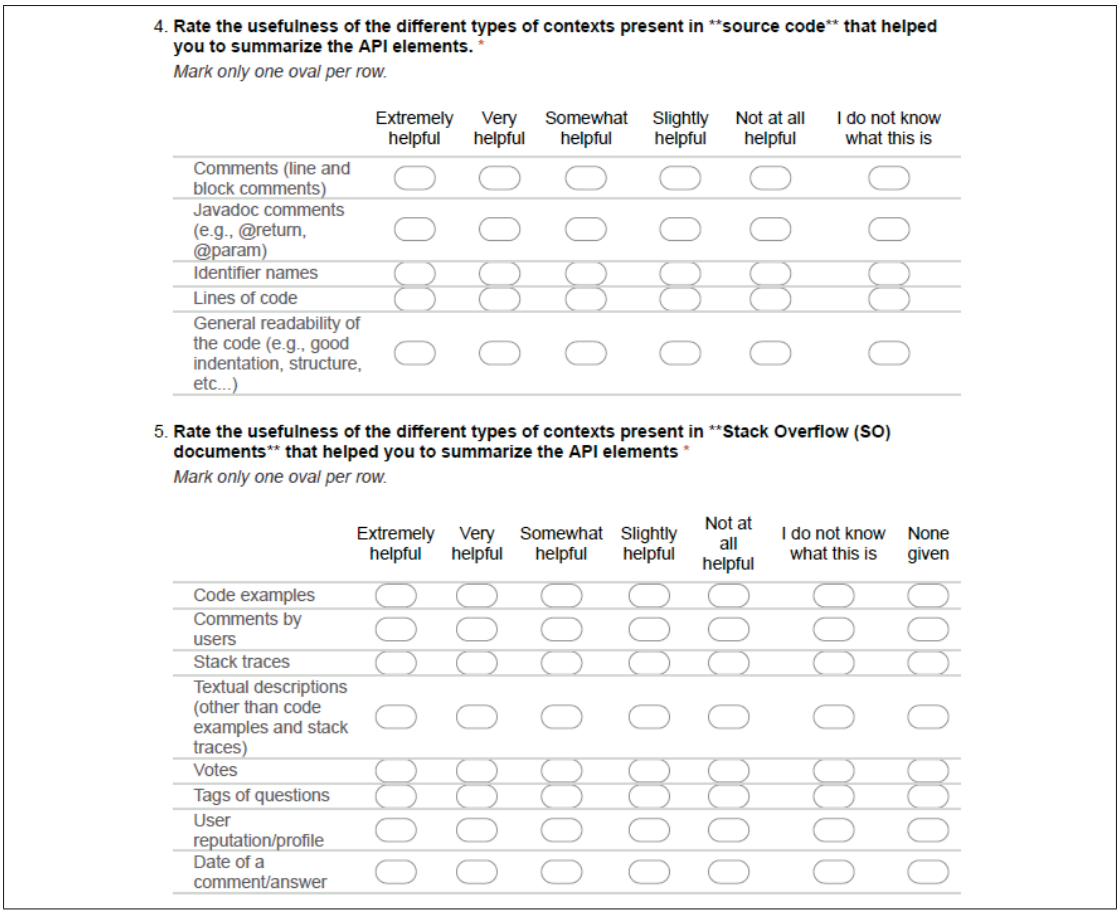
2. **Were you familiar with the code of the following projects or parts of these projects before this study?** *
   (By familiar, we mean you have seen or worked with this code, bug report, Stack Overflow document before and that knowledge helped you in your answers to the study.)
   Mark only one oval per row.

	Extremely familiar	Moderately familiar	Somewhat familiar	Slightly familiar	Not at all familiar
Eclipse					
Netbeans					
JMeter					
Tomcat					

3. **Rate the usefulness of each type of information with respect to how helpful they were to summarize the API elements in the study.** *
   Mark only one oval per row.

	Extremely helpful	Very helpful	Somewhat helpful	Slightly helpful	Not at all helpful
Stack Overflow					
Bug Reports					
Source Code					
Stack Overflow + Bug Reports + Source Code					

Figure-A II-8 Partie 1 : Post-questionnaire.



4. **Rate the usefulness of the different types of contexts present in **source code** that helped you to summarize the API elements.** *

Mark only one oval per row.

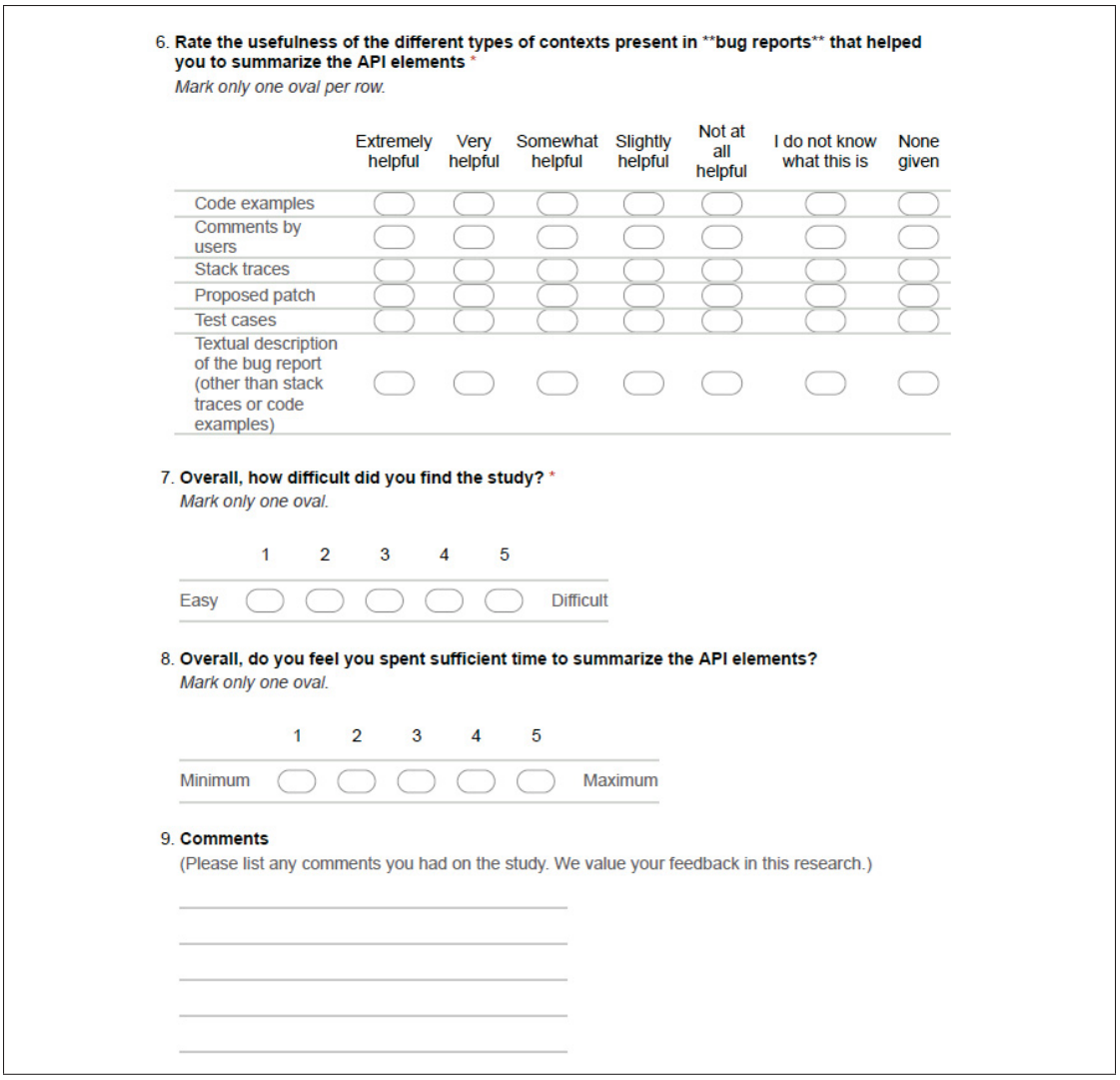
	Extremely helpful	Very helpful	Somewhat helpful	Slightly helpful	Not at all helpful	I do not know what this is
Comments (line and block comments)						
Javadoc comments (e.g., @return, @param)						
Identifier names						
Lines of code						
General readability of the code (e.g., good indentation, structure, etc...)						

5. **Rate the usefulness of the different types of contexts present in **Stack Overflow (SO) documents** that helped you to summarize the API elements** *

Mark only one oval per row.

	Extremely helpful	Very helpful	Somewhat helpful	Slightly helpful	Not at all helpful	I do not know what this is	None given
Code examples							
Comments by users							
Stack traces							
Textual descriptions (other than code examples and stack traces)							
Votes							
Tags of questions							
User reputation/profile							
Date of a comment/answer							

Figure-A II-9 Partie 2 : Post-questionnaire.



6. **Rate the usefulness of the different types of contexts present in **bug reports** that helped you to summarize the API elements** *
Mark only one oval per row.

	Extremely helpful	Very helpful	Somewhat helpful	Slightly helpful	Not at all helpful	I do not know what this is	None given
Code examples							
Comments by users							
Stack traces							
Proposed patch							
Test cases							
Textual description of the bug report (other than stack traces or code examples)							

7. **Overall, how difficult did you find the study?** *
Mark only one oval.

	1	2	3	4	5	
Easy						Difficult

8. **Overall, do you feel you spent sufficient time to summarize the API elements?**
Mark only one oval.

	1	2	3	4	5	
Minimum						Maximum

9. **Comments**
(Please list any comments you had on the study. We value your feedback in this research.)

Figure-A II-10 Partie 3 : Post-questionnaire.

ANNEXE III

PERFORMANCES DES DÉVELOPPEURS PROFESSIONNELS PAR L'UTILISATION DES DIFFÉRENTS TYPES DE SOURCES D'INFORMATIONS

1. Performances des développeurs professionnels par l'utilisation du code source lors de la synthèse de chaque élément de code.

Tableau-A III-1 Performances des développeurs professionnels : Précision, rappel et F-mesure en utilisant le code source pour chaque élément de code.

Élément de code	Précision	Rappel	F-mesure
Classe1	55.00	68.75	61.11
	25.58	68.75	37.29
	55.00	68.75	61.11
Méthode4	25.00	21.05	22.86
	12.77	33.33	18.46
	14.04	50.00	21.92
Méthode3	26.09	31.58	28.57
	10.00	5.56	7.14
	28.57	13.33	18.18
Méthode2	20.83	26.32	23.26
	42.86	50.00	46.15
	17.39	53.33	26.23
Méthode1	15.00	21.43	17.65
	22.22	33.33	26.67
	12.50	20.00	15.38
Classe2	90.91	71.43	80.00
	42.86	50.00	46.15
	17.24	47.62	25.32
Classe4	15.00	21.43	17.65
	14.04	50.00	21.92
	13.89	23.81	17.54
Classe3	10.00	5.56	7.14
	11.76	37.50	17.91
	17.54	47.62	25.64

Les performances des développeurs professionnels à synthétiser les éléments de code avec l'utilisation du code source sont représentées dans le tableau III-1, où les mesures de précision, rappel et F-mesure sont calculées pour chaque élément de code. Chaque élément de code a été synthétisé trois fois par l'utilisation d'une source d'informations. Dans ce cas, par l'utilisation du code source.

Voir le tableau 2.2 afin de retrouver les noms des classes et méthodes.

2. Performances des développeurs professionnels par l'utilisation du Stack Overflow lors de la synthèse de chaque élément de code

Tableau-A III-2 Performances des développeurs professionnels : Précision, rappel et F-mesure en utilisant le Stack Overflow pour chaque élément de code.

Élément de code	Précision	Rappel	F-mesure
Méthode2	63.64	43.75	51.85
	15.38	25.00	19.05
	63.64	43.75	51.85
Classe3	38.10	42.11	40.00
	33.33	16.67	22.22
	15.38	25.00	19.05
Classe2	22.22	21.05	21.62
	17.39	22.22	19.51
	10.17	40.00	16.22
Méthode1	38.10	42.11	40.00
	20.45	75.00	32.14
	18.75	20.00	19.35
Méthode3	25.00	14.29	18.18
	33.33	25.00	28.57
	11.86	46.67	18.92
Classe4	25.00	21.43	23.08
	20.93	75.00	32.73
	11.90	23.81	15.87
Classe1	71.43	35.71	47.62
	23.53	25.00	24.24
	13.95	28.57	18.75
Méthode4	17.39	22.22	19.51
	26.67	25.00	25.81
	12.50	23.81	16.39

Les performances des développeurs professionnels par l'utilisation du Stack Overflow sont représentées dans le tableau III-2, où les mesures de précision, rappel et F-mesure sont calculées pour chaque élément de code. Chaque élément de code a été synthétisé trois fois par l'utilisation du Stack Overflow.

Voir le tableau 2.2 afin de retrouver les noms des classes et méthodes.

3. Performances des développeurs professionnels par l'utilisation des rapports de bogues lors de la synthèse de chaque élément de code

Tableau-A III-3 Performances des développeurs professionnels : Précision, rappel et F-mesure en utilisant les rapports de bogues pour chaque élément de code.

Élément de code	Précision	Rappel	F-mesure
Classe3	11.11	12.50	11.76
	27.27	18.75	22.22
	11.76	12.50	12.12
Classe2	33.33	10.53	16.00
	19.05	22.22	20.51
	33.33	18.75	24.00
Méthode1	40.00	31.58	35.29
	14.29	22.22	17.39
	11.54	20.00	14.63
Classe1	33.33	10.53	16.00
	10.34	25.00	14.63
	21.05	26.67	23.53
Classe4	15.00	21.43	17.65
	14.81	33.33	20.51
	11.54	20.00	14.63
Méthode3	8.00	14.29	10.26
	10.71	25.00	15.00
	9.68	28.57	14.46
Méthode4	20.00	28.57	23.53
	15.38	25.00	19.05
	44.44	19.05	26.67
Méthode2	15.79	16.67	16.22
	57.14	25.00	34.78
	9.84	28.57	14.63

Les performances des développeurs professionnels par l'utilisation des rapports de bogues sont représentées dans le tableau III-3, où les mesures de précision, rappel et F-mesure sont calculées pour chaque élément de code. Chaque élément de code a été synthétisé trois fois par l'utilisation du rapport de bogues.

Voir le tableau 2.2 afin de retrouver les noms des classes et méthodes.

4. Performances des développeurs professionnels par l'utilisation de la combinaison des sources d'information lors de la synthèse de chaque élément de code

Tableau-A III-4 Performances des développeurs professionnels : Précision, rappel et F-mesure en utilisant la combinaison des sources pour chaque élément de code.

Élément de code	Précision	Rappel	F-mesure
Méthode4	4.17	6.25	5.00
	30.43	43.75	35.90
	38.89	43.75	41.18
Méthode1	7.87	52.63	13.70
	27.27	16.67	20.69
	36.36	25.00	29.63
Classe4	22.22	52.63	31.25
	9.68	16.67	12.24
	9.09	13.33	10.81
Classe3	8.26	52.63	14.29
	17.07	58.33	26.42
	17.86	33.33	23.26
Classe2	27.27	42.86	33.33
	14.63	50.00	22.64
	54.55	40.00	46.15
Méthode2	19.51	57.14	29.09
	16.67	58.33	25.93
	6.06	9.52	7.41
Méthode3	27.27	42.86	33.33
	40.00	25.00	30.77
	19.05	19.05	19.05
Classe1	8.33	16.67	11.11
	15.00	37.50	21.43
	6.25	9.52	7.55

Les performances des développeurs professionnels par l'utilisation de la combinaison des trois sources d'informations (code source, rapports de bogues et Stack Overflow) sont représentées dans le tableau III-4, où les mesures de précision, rappel et F-mesure sont calculées pour chaque élément de code.

Chaque élément de code a été synthétisé trois fois par l'utilisation de la combinaison des trois sources d'informations.

Voir le tableau 2.2 afin de retrouver les noms des classes et méthodes.

BIBLIOGRAPHIE

Bacchelli, A., Ponzanelli, L. & Lanza, M. (2012). Harnessing stack overflow for the ide. *Proceedings of the third international workshop on recommendation systems for software engineering*, pp. 26–30.

Baeza-Yates, R., Ribeiro-Neto, B. et al. (1999). *Modern information retrieval*. Addison-Wesley.

Basili, V. R., Caldiera, G. & Rombach, H. D. (1994). Experience factory. *Encyclopedia of software engineering*.

Carenini, G., Ng, R. T. & Zhou, X. (2007). Summarizing email conversations with clue words. *Proceedings of the 16th international conference on world wide web*, pp. 91–100.

Chen, M.-S., Han, J. & Yu, P. S. (1996). Data mining : an overview from a database perspective. *Ieee transactions on knowledge and data engineering*, 8(6), 866–883.

Dagenais, B. & Robillard, M. P. (2012). Recovering traceability links between an api and its learning resources. *Software engineering (icse), 2012 34th international conference on*, pp. 47–57.

Dit, B., Guerrouj, L., Poshyvanyk, D. & Antoniol, G. (2011). Can better identifier splitting techniques help feature location ? *Program comprehension (icpc), 2011 ieee 19th international conference on*, pp. 11–20.

Duala-Ekoko, E. & Robillard, M. P. (2012). Asking and answering questions about unfamiliar apis : An exploratory study. *Proceedings of the 34th international conference on software engineering*, pp. 266–276.

Grissom, R. J. & Kim, J. J. (2005). *Effect sizes for research : A broad practical approach* (éd. 2nd Edition). Lawrence Earlbaum Associates.

Guerrouj, L. (2013). *Context-aware source code identifier splitting and expansion for software maintenance*. (Thèse de doctorat, École Polytechnique de Montréal).

Guerrouj, L., Penta, M., Guéhéneuc, Y.-G. & Antoniol, G. (2014). An experimental investigation on the effects of context on source code identifiers splitting and expansion. *Empirical softw. engg.*, 19(6), 1706–1753.

Guerrouj, L., Bourque, D. & Rigby, P. C. (2015). Leveraging informal documentation to summarize classes and methods in context. *Software engineering (icse), 2015 ieee/acm 37th ieee international conference on*, 2, 639–642.

Haiduc, S., Aponte, J., Moreno, L. & Marcus, A. (2010). On the use of automated text summarization techniques for summarizing source code. *Proceedings of the 2010 17th working conference on reverse engineering*, (WCRE '10), 35–44. doi : 10.1109/WCRE.2010.13.

Holm, S. (1979). A simple sequentially rejective Bonferroni test procedure. *Scandinavian journal of statistics*, 6, 65–70.

Hung, H. J., O'Neill, R. T., Bauer, P. & Kohne, K. (1997). The behavior of the p-value when the alternative hypothesis is true. *Biometrics*, 11–22.

Jones, K. S. (2007). Automatic summarising : The state of the art. *Information processing & management*, 43(6), 1449–1481.

Linares-Vásquez, M., Bavota, G., Bernal-Cárdenas, C., Di Penta, M., Oliveto, R. & Poshyvanyk, D. (2013). Api change and fault proneness : a threat to the success of android apps. *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pp. 477–487.

McBurney, P. W. & McMillan, C. (2014). Automatic documentation generation via source code summarization of method context. *Proceedings of the 22nd international conference on program comprehension*, (ICPC 2014), 279–290. doi : 10.1145/2597008.2597149.

McBurney, P. W. & McMillan, C. (2016). Automatic source code summarization of context for java methods. *Ieee transactions on software engineering*, 42(2), 103–119.

McGill, R., Tukey, J. W. & Larsen, W. A. (1978). Variations of box plots. *The american statistician*, 32(1), 12–16.

Moreno, L. & Aponte, J. (2012). On the analysis of human and automatic summaries of source code. *Clei electronic journal*, 15(2), 2–2.

Moreno, L., Aponte, J., Sridhara, G., Marcus, A., Pollock, L. & Vijay-Shanker, K. (2013a, May). Automatic generation of natural language summaries for java classes. *Program comprehension (icpc), 2013 ieee 21st international conference on*, pp. 23–32.

Moreno, L., Marcus, A., Pollock, L. & Vijay-Shanker, K. (2013b). Jsummarizer : An automatic generator of natural language summaries for java classes. *Program comprehension (icpc), 2013 ieee 21st international conference on*, pp. 230–232.

Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A. & Canfora, G. (2014). Automatic generation of release notes. *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, pp. 484–495.

Moreno, L., Bavota, G., Di Penta, M., Oliveto, R. & Marcus, A. (2015). How can i use this method ? *Proceedings of the 37th international conference on software engineering - volume 1*, (ICSE '15), 880–890. Repéré à http://dl.acm.org/citation.cfm?id=2818754.2818860.

Murphy, G. C. (1996). *Lightweight structural summarization as an aid to software evolution*. (Thèse de doctorat).

Petrosyan, G., Robillard, M. P. & De Mori, R. (2015). Discovering information explaining api types using text classification. *Proceedings of the 37th international conference on software engineering-volume 1*, pp. 869–879.

Ponzanelli, L., Mocci, A. & Lanza, M. (2015). Summarizing complex development artifacts by mining heterogeneous data. *Proceedings of the 12th working conference on mining software repositories*, pp. 401–405.

Rastkar, S. (2010). Summarizing software concerns. *Software engineering, 2010 acm/ieee 32nd international conference on*, 2, 527–528.

Rastkar, S. & Murphy, G. C. (2013). Why did this code change ? *Proceedings of the 2013 international conference on software engineering*, pp. 1193–1196.

Rastkar, S., Murphy, G. C. & Murray, G. (2010). Summarizing software artifacts : a case study of bug reports. *Proceedings of the 32nd acm/ieee international conference on software engineering-volume 1*, pp. 505–514.

Rastkar, S., Murphy, G. C. & Bradley, A. W. (2011). Generating natural language summaries for crosscutting source code concerns. *Software maintenance (icsm), 2011 27th ieee international conference on*, pp. 103–112.

Ricca, F., Di Penta, M., Torchiano, M., Tonella, P. & Ceccato, M. (2010). How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes : A series of four experiments. *Ieee trans. software eng.*, 36(1), 96-118.

Robillard, M. P. & Chhetri, Y. B. (2015). Recommending reference api documentation. *Empirical software engineering*, 20(6), 1558–1586.

Robillard, M. P. & Deline, R. (2011). A field study of api learning obstacles. *Empirical software engineering*, 16(6), 703–732.

Sridhara, G., Hill, E., Muppaneni, D., Pollock, L. & Vijay-Shanker, K. (2010). Towards automatically generating summary comments for java methods. *Proceedings of the ieee/acm international conference on automated software engineering*, (ASE '10), 43–52. doi : 10.1145/1858996.1859006.

Subramanian, S., Inozemtseva, L. & Holmes, R. (2014). Live api documentation. *Proceedings of the 36th international conference on software engineering*, pp. 643–652.

Sun, X., Geng, Q., Lo, D., Duan, Y., Liu, X. & Li, B. (2016). Code comment quality analysis and improvement recommendation : An automated approach. *International journal of software engineering and knowledge engineering*, 26(06), 981–1000.

Tian, Y., Lo, D., Xia, X. & Sun, C. (2015). Automated prediction of bug report priority using multi-factor analysis. *Empirical software engineering*, 20(5), 1354–1383.

Treude, C. & Robillard, M. P. (2016). Augmenting api documentation with insights from stack overflow. *Proceedings of the 38th international conference on software engineering*, pp. 392–403.

Uddin, G. & Robillard, M. P. (2015). How api documentation fails. *Ieee software*, 32(4), 68–75.

Újházi, B., Ferenc, R., Poshyvanyk, D. & Gyimóthy, T. (2010). New conceptual coupling and cohesion metrics for object-oriented systems. *Source code analysis and manipulation (scam), 2010 10th ieee working conference on*, pp. 33–42.

Venkatesh, P. K., Wang, S., Zhang, F., Zou, Y. & Hassan, A. E. (2016). What do client developers concern when using web apis ? an empirical study on developer forums and stack overflow. *Web services (icws), 2016 ieee international conference on*, pp. 131–138.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. (2000). *Experimentation in software engineering - an introduction*. Kluwer Academic Publishers.

Xia, X., Lo, D., Shihab, E., Wang, X. & Zhou, B. (2015). Automatic, high accuracy prediction of reopened bugs. *Automated software engineering*, 22(1), 75–109.

Xia, X., Lo, D., Shihab, E. & Wang, X. (2016). Automated bug report field reassignment and refinement prediction. *Ieee transactions on reliability*, 65(3), 1094–1113.

Xie, T., Pei, J. & Hassan, A. E. (2007). Mining software engineering data. *Companion to the proceedings of the 29th international conference on software engineering*, pp. 172–173.

Ying, A. T. & Robillard, M. P. (2013). Code fragment summarization. *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pp. 655–658.

Ying, A. T. & Robillard, M. P. (2014). Selection and presentation practices for code example summarization. *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, pp. 460–471.

Zibran, M. F., Eishita, F. Z. & Roy, C. K. (2011). Useful, but usable ? factors affecting the usability of apis. *Reverse engineering (wcre), 2011 18th working conference on*, pp. 151–155.