2025

SNAIL Report

Baromètre des pratiques de développement logiciel en Fédération Wallonie-Bruxelles

Aline Boulanger, Yassine Bouncer, Louise Delpierre Stasser, Xavier Devroey, Antonin Lesceux, Jérôme Maquoi, Kévin Schweitzer, Benoît Vanderose, Lucas Vervloessem

Avant-propos

Le développement logiciel est un domaine en constante évolution, porté par des innovations technologiques et des pratiques professionnelles qui évoluent rapidement. Dans ce contexte, comprendre l'état actuel des pratiques de qualité et de test logiciel est essentiel, tant pour les entreprises que pour les institutions académiques. C'est dans cette optique que nous avons mené une enquête auprès des acteurs du développement logiciel en Fédération Wallonie-Bruxelles.

Cette initiative est née d'une collaboration entre des membres de la SNAIL team et les étudiants du master en software engineering, dans le cadre du cours « Test et Qualité logicielle ». L'objectif était double : **dresser un état des lieux des pratiques en vigueur** et **renforcer le lien entre la formation académique et les réalités industrielles**. Pour ce faire, nous avons conçu un questionnaire inspiré des grands rapports internationaux (JetBrains Developer Ecosystem, State of DevOps, GitHub Octoverse) et adapté aux spécificités locales.

L'enquête a été diffusée auprès d'organisations de tailles et de secteurs variés, afin de recueillir des données représentatives. Les résultats présentés dans ce rapport offrent un panorama des pratiques actuelles : adoption des méthodologies Agile, organisation des équipes, outils utilisés, stratégies de test, intégration de l'IA, automatisation des processus, et perception globale de la qualité logicielle.

Au-delà des chiffres, ce rapport met en lumière les **forces**, les **défis** et les **opportunités** qui se dessinent pour les entreprises et les professionnels du développement logiciel en Fédération Wallonie-Bruxelles. Nous espérons que ces enseignements contribueront à nourrir la réflexion, à améliorer les pratiques et à renforcer la collaboration entre les mondes académique et industriel.

Nous remercions chaleureusement toutes les personnes et organisations qui ont participé à cette enquête et qui ont rendu ce travail possible.

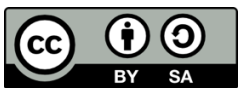


Table des matières

Avant-propos 1

Table des matières 2

Introduction 3

Profil des organisations 3

Méthodologies et organisation en équipe 4

Qualité et collaboration 5

Tests et automatisation 7

Documentation et standards 9

Intelligence artificielle 10

Sécurité et dépendances 11

Technologies et plateformes 13

Expérience développeur 15

Données démographiques 16

Méthodologie 17

Conclusion 19

Introduction

Ce rapport présente les résultats détaillés de l'enquête menée entre avril et juillet 2025 sur **les pratiques de qualité et de test logiciel au sein des organisations de la Fédération Wallonie-Bruxelles** (PME, ETI, grandes entreprises et secteur public). Cette étude, qui rassemble **52 répondants**, s'inscrit dans une démarche visant à mieux comprendre les méthodes, outils et valeurs qui structurent le développement logiciel dans notre région, afin d'alimenter la réflexion académique et professionnelle sur l'amélioration continue des pratiques. Elle met en lumière les forces, les défis et les opportunités des processus de développement logiciel.

Le rapport est organisé selon les **grandes thématiques** du développement logiciel. Pour chaque thématique, quelques points clés sont mis en avant, suivis de résultats détaillés relatifs aux questions qui s'y rapportent.

Public cible

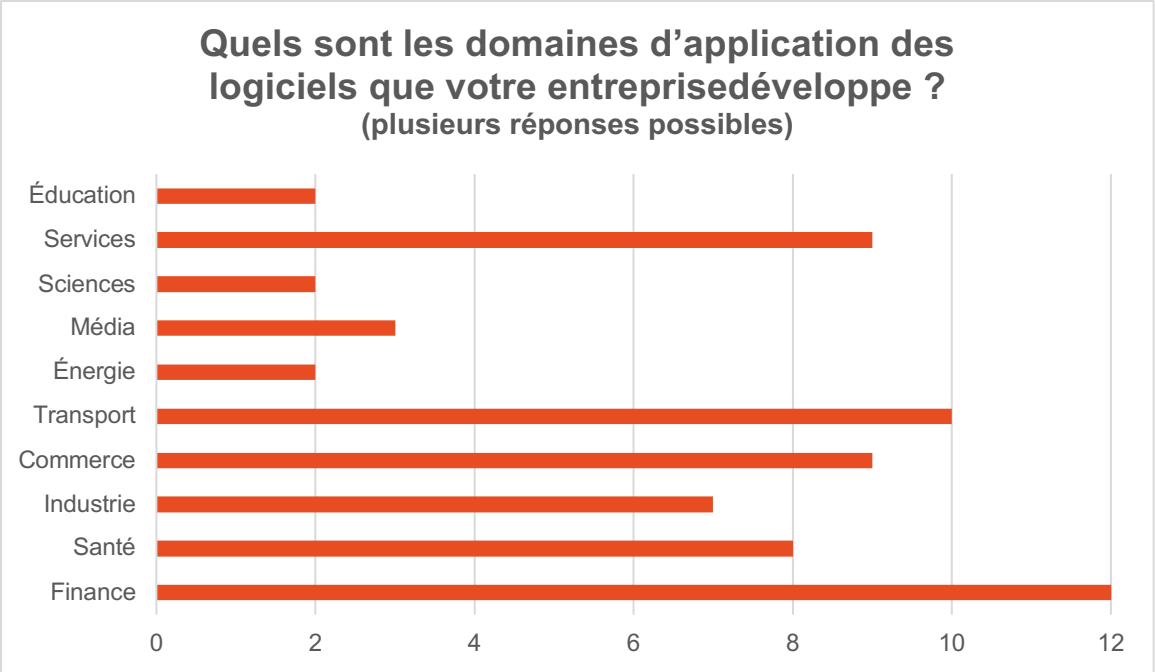
Ce rapport s'adresse aux **professionnels du développement logiciel**, aux **responsables qualité**, aux **architectes**, ainsi qu'aux **enseignants et chercheurs** souhaitant adapter leurs contenus pédagogiques aux réalités du terrain.

Profil des organisations

Points clés

- **Taille** : Majoritairement des ETI (41 %) et PME (31 %).
- **Secteurs** : Principalement privé (75 %), avec une forte représentation des services aux citoyens dans le public.
- **Domaines** : Finance (36 %), Transport (30 %), Commerce (27 %), Santé (24 %), Industrie (21 %)

Les personnes interrogées travaillent dans toute la fédération Wallonie-Bruxelles. 3 répondant·e·s vivent en Belgique, mais travaillent au Luxembourg. La majorité travaille dans une ETI (41 %) ou une PME (31 %), le reste se distribuant entre les micro-entreprises (14 %) et les grandes entreprises (12 %). 3 personnes interrogées ont indiqué que leur équipe de développement se situe dans un pays différent de celui du siège de leur organisation, ce qui montre que certaines entreprises délocalisent leur développement.



Les personnes interrogées travaillent principalement dans des entreprises du secteur privé (75 %) et développent du logiciel pour différents domaines d'application repris ci-dessus. Parmi les 25 % des personnes travaillant dans le secteur public, une large majorité (70 %) développe des logiciels à destination des citoyens dans des administrations centrales (80 %).

Méthodologies et organisation en équipe

Points clés

- **Méthodes de développement** : 69 % des équipes adoptent une approche Agile, souvent via Scrum ou ses variantes à l'échelle.
- **Organisation** : 41 % combinent des équipes spécialisées et polyvalentes.
- **Perception** : 79 % jugent leur méthode de développement positive pour la qualité.

24% des personnes interrogées indiquent que leur organisation est plutôt organisée en équipes spécialisées (équipes dédiées à l'assurance qualité, à l'analyse fonctionnelle, etc.), 36% indiquent être organisée autour d'équipes polyvalentes et une majorité (41%) indiquent que leur organisation combine les deux. Une majorité des réponses indique qu'une équipe de développement interagit directement avec 2 à 3 autres équipes (40 %), plus de 5 équipes (28 %) ou une seule autre équipe (15 %).

De manière générale, on observe une **grande disparité au niveau des approches de développement** suivies par les équipes. La majorité (69 %) travaille en suivant une approche Agile et seulement 5% indiquent suivre une approche prédictive de

type Waterfall ou modèle en V. Le reste (26 %) indique avoir une approche variable en fonction des projets ou produits développés. Parmi les approches Agile, *Scrum@Scale* (43 %) et *Enterprise Scrum* (37 %) sont les plus répandues. 14 % indiquent suivre une approche Agile Portfolio Management et 9 % indiquent utiliser le Scaled Agile Framework. 17 % des personnes interrogées indiquent suivre un autre type d'approche Agile, parmi lesquelles des approches personnalisées sur base de l'expérience de l'équipe ou propres à l'entreprise (3 répondant·e·s) et une approche Agile propre au développement d'appareils médicaux (1 répondant·e).

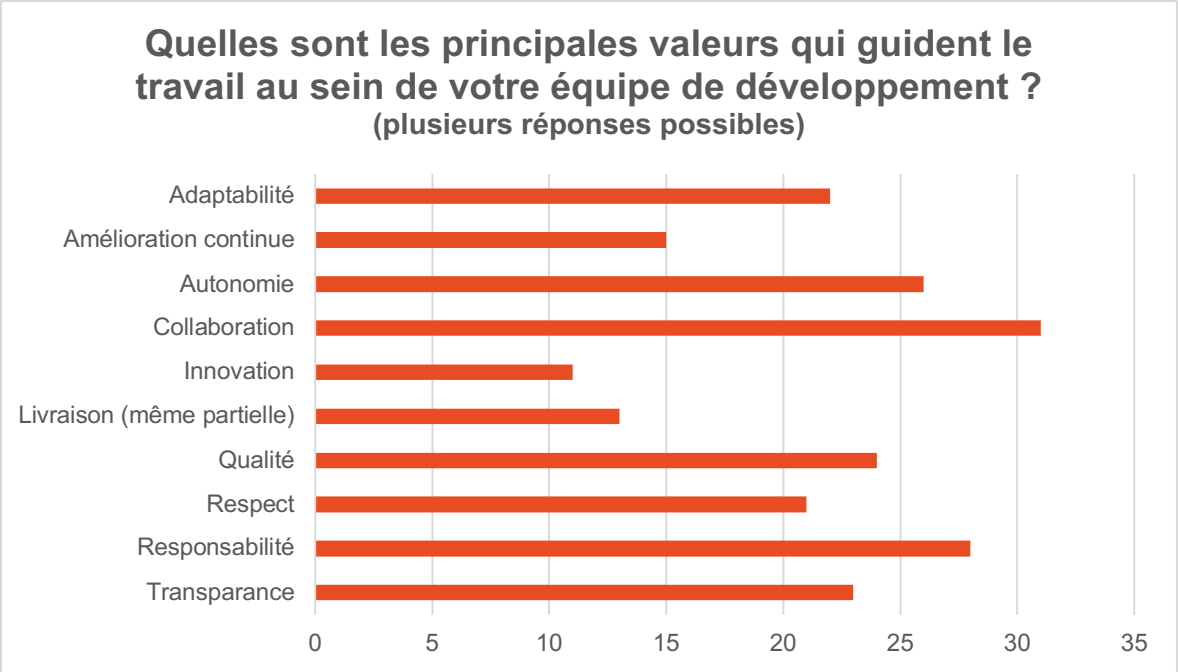
La majorité des répondant·e·s jugent que leur approche de développement a un impact plutôt positif (61 %) ou très positif (18 %) sur la qualité du code et de la collaboration. 13 % indiquent ne voir aucun impact ou ne pas savoir, et seuls 3 % indiquent un impact négatif, avec une méthode induisant des difficultés dans le travail. Enfin, 5 % préfèrent ne pas répondre.

Qualité et collaboration

Points clés

- **Valeurs clés** : Collaboration (84 %), responsabilité (76 %), qualité (65 %) et autonomie (70 %).
- **Pratiques de collaboration** : Usage massif d'outils comme Jira (84 %), Confluence (82 %), et Git (72 %).
- **Revue de code** : Pratique courante (68 % à chaque PR), perçue comme utile voire formatrice.

Le graphique suivant présente les principales valeurs qui guident le travail au sein des équipes de développement. On observe que les valeurs relatives à l'équipe et aux personnes sont prédominantes avec la **collaboration**, permettant le travail ensemble et le partage de connaissances pour atteindre les objectifs communs, pour 84 % des répondant·e·s, suivie de la **responsabilisation** par rapport aux tâches et à la qualité du travail fourni pour 76 % des répondant·e·s et de l'**autonomie,** permettant à chacun et chacune de prendre des décisions et de gérer son travail pour 70 % des répondant·e·s. La **qualité** du code, des produits et des processus n'arrive qu'en quatrième position avec 65 % des répondant·e·s. L'innovation, afin d'encourager la créativité et l'expérimentation pour améliorer les solutions, arrive en dernier avec seulement 30% des répondant·e·s indiquant qu'il s'agit d'une valeur principale de l'équipe.



Les répondant·e·s indiquent que la collaboration sur le développement de fonctionnalités complexes se fait via des **outils de gestion de projet** (88 %) tels que JIRA (84 %), Azure Boards (9 %), GitHub issues (9%), Asana (3 %), etc.

Parmi les impacts de ces produits sur la qualité logicielle, les répondant·e·s citent la **traçabilité** (« […] *la certitude de lier correctement des tâches avec des Pull Request, la traçabilité ISO 27100 grâce au numéro d'issue-ticket-PR-commits* »), la **priorisation** (« […] priorisation des tâches en fonction des dépendances. ») et le **suivi** des différentes tâches (« *Un suivi régulier dans JIRA permet de s'assurer du planning* […] », « *Il n'est pas rare que durant nos sprint rétrospective meetings, on se rende compte qu'on se soit un peu égaré lors de notre développement* […] »). On retrouve aussi un **soutien à la collaboration** (« […] *Entre-aide possible en découpant une tâche complexe en sous tâche plus simple* ») et à l'**automatisation** (« *L'utilisation de workflows basé sur Jira a permis d'augmenter la qualité du logiciel produit en systématisant un certain nombre de processus : code review, génération automatique des nouvelles "releases" (embarquement des tickets tagués), génération automatique des "release notes"* »).

Les défis liés à l'utilisation d'outils de gestion de projets mentionnés sont la **complexité** qui demande une phase d'adaptation (« *l'outil offre énormément d'éléments mais requiert une phase d'adaptation, de mise en place de règle/automatisme* »), la nécessité de pouvoir **adapter l'outils** aux besoins de l'équipe (« *JIRA est installé de la même manière chez toutes les équipes et il assez difficile de l'adapter à nos besoins spécifiques car nous n'avons pas tous les droits dessus.* »), la **mise à jour de la documentation** (« [Il est difficile de] *maintenir la documentation (les "how to", procédures, etc) à jour surtout quand ce sont des procédures peu utilisées* ») et le respect des conventions décrites (« *Définir une convention commune pour toutes les équipes et qu'elle soit respectée*. »). Enfin,

certain·e·s participant·e·s ont indiqué des soucis d'utilisabilité (« *Interface peu attractive et peu pratique.* », « *Updates réguliers et non planifiés d'Atlassian.* », « *Lenteur des interfaces Jira.* »).

La collaboration repose également sur des **outils de documentation collaborative** (82 %) tels que Confluence, Notion, etc. ; une **gestion des branches de fonctionnalités** spécifiques dans le gestionnaire de version tels que Git Flow, GitHub Flow, etc. (77 %) ; du ***pair programming*** ou du ***peer reveiwing*** (35 %) ; et engin, la mise en place de ***community of practices*** (21 %).

Tests et automatisation

Points clés

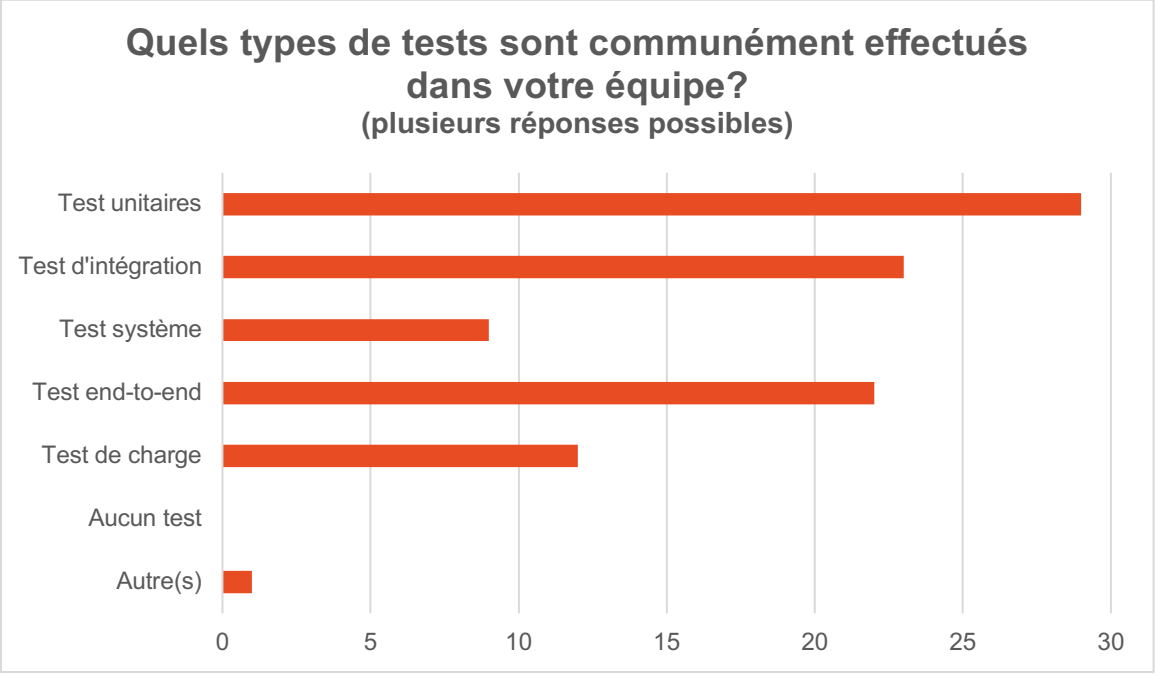
- **Tests** : 94 % pratiquent les tests unitaires, 75 % les tests d'intégration et 71 % les tests end-to-end. 71 % ont une stratégie de test organisationnelle.
- **CI/CD** : 66 % utilisent l'intégration continue et 81% automatisent leurs déploiements. Jenkins, GitLab CI et Azure Pipelines sont les outils les plus utilisés.
- **Impact** : Les pipelines CI/CD améliorent la rapidité et la fiabilité des déploiements.

Une majorité des personnes sondées indique que l'intégration continue et le déploiement sont automatisés dans une pipeline à l'aide d'outils tels que Jenkins, GitLab CI, Azure Pipelines et TeamCity. Au niveau des différentes étapes, ces pipelines incluent de l'analyse de code (52%), l'exécution de test unitaires (78%) et d'intégration (59%), le déploiement automatisé en environnement de test (74%) et en environnement de production (30%). Les bénéfices indiqués par les répondants·es incluent l'accélération des activités de développement et de déploiement (« *passée de 10 heures à 20 minutes grâce à l'amélioration des outils CI/CD* »), ainsi qu'une amélioration de la qualité grâce à l'exécution centralisée et systématique des tests. Deux répondants·es notent néanmoins un problème de volume pour certains dépôts de code, pour lesquels il faut entre une et plusieurs heures pour exécuter le *build* : « *certains repo sont devenus trop gros et demandent du temps (parfois +1h) avant d'arriver au bout de la pipeline.* » L'une des deux réponses précise que l'entièreté de la pipeline s'exécute totalement durant la nuit, ce qui est une stratégie couramment adoptée pour éviter des temps d'attente trop longs durant les activités de développement durant la journée.

Au niveau des défis rencontrés lors de la mise en place d'une pipeline CI/CD, les personnes sondées indiquent un manque de standardisation des différentes plateformes de CI/CD qui ont chacune leurs propres subtilités. Il est aussi fait mention des coûts si la pipeline est externalisée vers un service tiers. La majorité des réponses indiquent un manque de flexibilité de l'organisation quant aux possibilités

d’adapter une pipeline à un contexte spécifique : « *utiliser un outil qu'on nous a imposé sans pouvoir l'adapter entièrement à nos besoins* », « *incompatible avec les méthodes génériques du département IT.* » Enfin, un·e répondant·e évoque un problème de culture et d’« *évolution des mentalités vers la compréhension des impacts d'un vrai CI/CD*. »

En ce qui concerne les tests, 71% des répondants·es indiquent qu’il existe une stratégie de test définie au niveau organisationnel, 13% indiquent que ce n’est pas le cas et 16% indiquent ne pas savoir. La couverture de code (i.e. le pourcentage de lignes de code exécutées par les tests) est la principale mesure utilisée avec des seuils à atteindre compris entre 50% et 90 %, voire 100%. À noter que le seuil minimal généralement admis comme « bon » se situe entre 80 et 90 %. Certaines stratégies observées dans des projets open source comme XWiki prônent une « non-diminution » de la couverture des tests entre deux versions du code. Une majorité des personnes sondées (79 %) indiquent qu’une partie des tests sont exécutés manuellement avec un pourcentage de tests manuels variant fortement entre 5 % et 100 %. Les tests sont perçus plutôt positivement et permettent de « *trouver des bugs* », mais aussi de « *la non-régression des fonctionnalités* ». Un·e répondant·e indique que les activités de test ont permis de clarifier les contraintes métier et de les coller au mieux. Enfin, un·e répondant·e indique que les tests unitaires ont eu un impact sur les activités de *refactoring*, tendant à montrer que la testabilité du code est essentielle pour écrire des tests unitaires. En plus des tests unitaires, les tests se font à différents niveaux, comme indiqué ci-dessous :



Documentation et standards

Points clés

- **Guides de développement** : 44 % disposent d'un guide partiel, 26 % d'un guide complet.
- **Documentation technique** : Centralisée dans 82 % des cas.
- **Analyse de code** : SonarQube est utilisé dans 57 % des cas, souvent intégré aux pipelines.

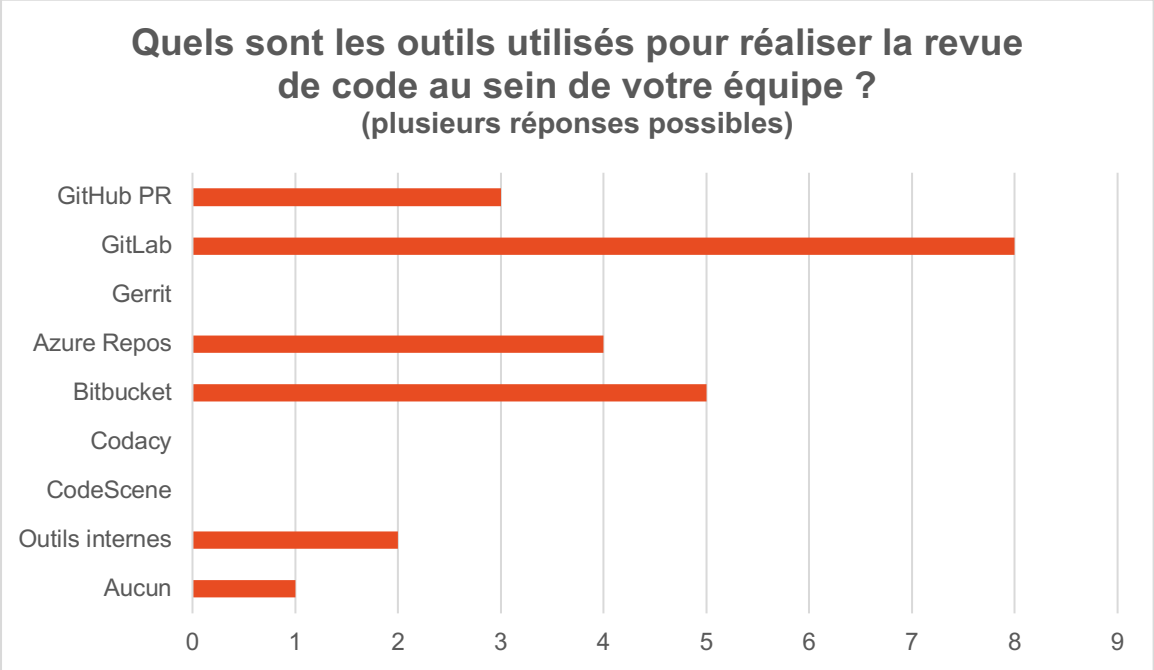
Une majorité des répondant·e·s (38 %) indique **ne pas avoir de guide de contribution** pour leur projet ou leur base de code. 29 % indiquent qu'un tel guide existe, mais qu'il n'est pas formalisé et souvent mis à jour, tandis que 18 % indiquent que ce guide est formalisé, mais pas à jour et donc peu utilisé. Seuls 18 % indiquent qu'un guide **complet et à jour** existe, avec des instructions détaillées sur la manière de contribuer au projet.

La **documentation technique** (documentation des APIs, documents d'architecture, spécification technique, etc.) est majoritairement **centralisée** dans un outil de documentation tel que Confluence, Notion ou un wiki interne (82 % des répondant·e·s). Pour 50 % des répondant·e·s, la documentation est stockée directement dans le référentiel de code (par exemple, dans des fichiers Markdown sur le dépôt Git). Pour 15 %, la documentation est dispersée dans différents outils, non centralisée, et seulement 9 % indiquent ne pas avoir de documentation technique structurée.

Au niveau du **code**, la majorité des personnes interrogées indique disposer d'un **guide de programmation** ou d'une charte de développement, soit **partiel** (44 %) ou **complet** (26 %). 18 % indiquent également disposer d'un tel guide, fondé sur des normes internationales pour le C# et le C (normes ISO et IEC). 26 % indiquent ne pas avoir de guide formel de programmation. En ce qui concerne la **documentation du code**, la majorité des personnes interrogées indique ne pas avoir de norme formelle pour la documentation du code (53 %), contre 32 % qui indiquent que ces normes ne sont pas systématiquement appliquées, et seulement 18 % suivent des normes de documentation telles que Javadoc, Docstring, Sphinx, etc.

La vérification de l'**application des bonnes pratiques** et standards de codage se fait principalement par la **revue de code régulière** (62 % des répondant·e·s), l'utilisation d'**outils d'analyse statique** de code ou de **linters** intégrés au processus de CI/CD (56 %) et des **formations internes** ou des **sessions de partage** de bonnes pratiques (44 %). 21 % indiquent ne pas avoir de processus formalisé pour fournir ce type de garanties, et 6 % indiquent passer par une équipe dédiée au contrôle qualité et aux tests.

La **revue de code** peut se faire à différents moments. 68% des répondant·e·s indiquent qu'elle se pratique à chaque commit ou Pull Request. 18% indiquent qu'elle se déroule selon les besoins, et 18% qu'elle se déroule rarement. 7% indiquent que la revue de code se fait avant la mise en production et 11% indiquent qu'elle ne se pratique jamais. La majorité des personnes interrogées (41 %) indique que la revue de code est *utile, mais parfois perçue comme un frein à la rapidité*. 33 % indiquent qu'*elle est une opportunité d'apprentissage, d'amélioration et de collaboration*, 15 % qu'*elle est perçue comme une formalité n'apportant pas de valeur,* et 7 % qu'*elle est rarement faite ou pas prise au sérieux, ce qui nuit à la qualité du code*. Ci-après, la liste des outils utilisés pour la revue de code.



Intelligence artificielle

Points clés

- **Adoption** : 71 % utilisent l'IA, principalement pour la génération de code (45 %) et l'analyse de qualité (23 %).
- **Outils** : Copilot (64 %) et ChatGPT (59 %) sont les plus populaires.
- **Défis** : Manque de formation (68 %) et coût (42 %).
- **Opportunités** : Productivité (67 %), qualité du code (38 %), innovation (38 %).

L'IA générative est utilisée par 71 % des répondants·es dans le cadre de leurs activités de développement logiciel : 19 % indiquent l'utiliser pour la **génération de tests** ; 45 % indiquent l'utiliser pour la **génération de code** ; 23 % l'utilisent comme **outils d'analyse** qui aide à détecter les erreurs de code, les vulnérabilités et à suggérer des améliorations ; 10% indiquent l'utiliser pour o**ptimiser les**

performances de code ; 19% l'utilisent comme **aide à la rédaction ou à la génération automatique de la documentation** du code ; 16% l'utilisent pour orienter les choix technologiques ou les décisions de conception de code ; et enfin, 22% indiquent l'utiliser pour d'autres cas, tels que le debugging, comme assistant sans processus généralisé au sein des équipes, comme préparation de réponses au helpdesk, ou plus généralement pour expérimenter et voir quelles sont les possibilités.

ChatGPT est utilisé par 59 % des répondants·es, 63 % indiquent utiliser Copilot, 22 % utilisent Gemini, 14 % utilisent Mistral et 14 % utilisent l'assistant AI de JetBrains. Un·e répondant·e indique utiliser un outil basé sur une IA locale (Ollama) et un·e répondant·e indique utiliser un outil développé en interne.

Une majorité des réponses indique que l'IA est perçue comme ayant un **impact plutôt positif sur la qualité** du logiciel développé (67 %), 12% indiquent qu'il n'y a **pas d'impact ou ne pas savoir** et seuls 4 % indiquent que l'IA introduit des difficultés dans le travail. L'IA améliore la **productivité** (67 % des répondants·es) et la **qualité du code** (38 %), **facilite l'innovation** en permettant d'explorer de nouvelles idées et approches techniques (38 %), et **optimise les processus** de développement en améliorant la gestion des ressources et en automatisant les tâches (19 %).

Les défis rencontrés reprennent **le manque de formation** des équipes de développement pour tirer pleinement parti de ces outils IA (69 %), le **coût** d'acquisition et de maintenance des solutions IA (42 %), des **difficultés d'intégration** aux outils existants (32 %) et **d'interprétabilité** permettant de comprendre certaines décisions (32 %), ainsi qu'un **manque de contrôle humain** (26 %). Un·e répondant·e indique également une « *crainte de partage du code contenant de la logique business* », qui se traduirait finalement par des **fuites de code** vers l'extérieur.

Sécurité et dépendances

Points clés

- **Stratégies de sécurité** : Présentes dans 54 % des organisations.
- **Gestion des dépendances** : 56 % utilisent un dépôt local, 44 % pratiquent le versioning figé.
- **Surveillance** : Elastic Stack, OpenTelemetry et Sentry sont les outils les plus utilisés.

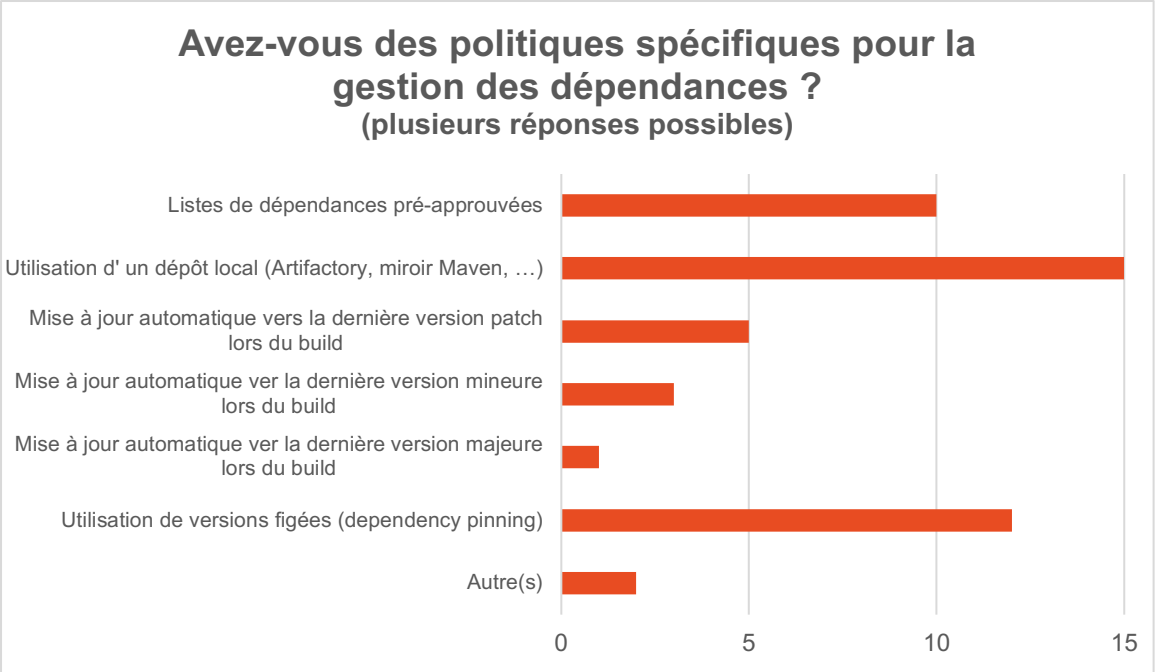
Les aspects de sécurité dans le cycle de vie logiciel interviennent à divers endroits. Au niveau du développement et de la maintenance, plusieurs répondants·es indiquent utiliser de l'analyse statique de code pour améliorer la qualité du code, mais aussi pour des aspects de sécurité : « *éviter des erreurs classiques comme des*

clés/tokens laissé en commentaire » ; « [l'analyse de code] *nous a déjà évité des problèmes de sécurité* » ; « [l'analyse de code] *concerne surtout la sécurité* ». 66% des répondants·es indiquent utiliser de l'analyse de code, avec une large majorité utilisant SonarQube, un outil d'analyse de code généraliste (57 %), et un·e répondant·e utilisant Snyk, un outil d'analyse de code spécialisé dans la sécurité. Ces outils sont lancés à différents moments : lors des builds (52 %), à chaque commit (44 %), à chaque release d'une version mineure ou patch (28 %) ou majeure (24 %), ou de manière périodique (8 %).

Une fois en production, des données télémétriques peuvent être collectées à des fins d'audit et de sécurité (45 % des répondants·es) ou pour repérer des erreurs et des exceptions (82 % des répondants·es). Les autres cas d'utilisation incluent la collecte de données de performance (50 %) et l'utilisation des fonctionnalités (36 %). 14 % des répondants·es indiquent ne pas collecter de données télémétriques. Les outils les plus populaires pour la télémétrie sont l'Elastic stack (28 %), OpenTelemetry (28 %), Application Insights (17 %) et DataDog (11 %).

Concernant la gouvernance, 54 % des réponses indiquent qu'une stratégie de gestion de la sécurité des produits logiciels est définie au niveau de l'organisation et appliquée, 22 % indiquent qu'une stratégie est définie mais pas ou mal appliquée, et seuls 14 % indiquent qu'aucune stratégie n'est définie. Le reste des répondants·es ne sachant pas, ou préférant ne pas répondre. Au niveau de l'équipe de développement, seuls 39 % indiquent avoir une stratégie définie et appliquée et 22 % indiquent que celle-ci est mal ou pas appliquée. 29 % des répondant·e·s indiquent également ne pas avoir de stratégie formellement définie, mais que des pratiques spécifiques en matière de sécurité sont en place. Le reste ne sait pas ou préfère ne pas répondre.

Enfin, concernant la gestion des dépendances, la majorité des répondants·es indiquent que les équipes de développement utilisent des dépôts locaux contenant les dépendances autorisées au niveau de l'organisation et/ou utiliser des versions figées :



Les répondants·es indiquent utiliser différentes stratégies pour faire de la veille de leurs dépendances et détecter des failles de sécurité : l'« *analyse des vulnérabilité via OWASP* » ou via « *des CVE* » ; un « *bot qui crée des merge requests dès qu'une dépendance doit être mise à jour pour des raisons de sécurité* » ; ou une gestion où « *chaque dépendance fait l'objet d'un mini-audit* » avant d'être ajoutée à un dépôt local. Concernant la détection et la correction des vulnérabilités dans les dépendances après la mise en production, il n'existe pas de stratégie majoritaire. Les répondants·es indiquent se reposer sur l'architecture en microservices pour isoler les services potentiellement impactés, une prise en charge la plus rapide possible (« *-24 h après la publication d'une CVE* ») ou externaliser ces aspects à une équipe dédiée à la cybersécurité. Seuls 36 % des répondants·es indiquent disposer de politiques spécifiques pour la gestion des licences de dépendance, telles que l'utilisation de dépendances préapprouvées (24 %) et l'interdiction de dépendances sous licence contaminante (20 %).

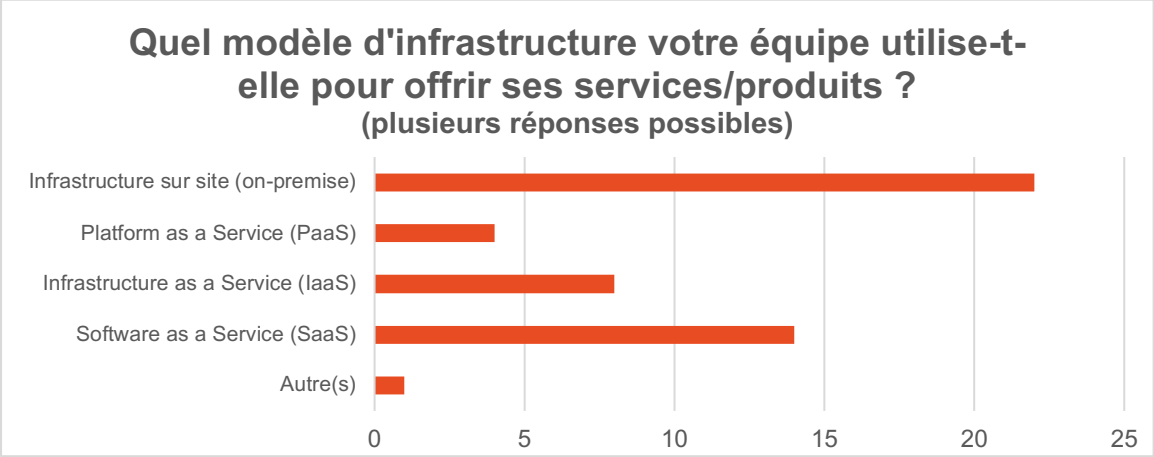
Technologies et plateformes

Points clés

- **Types de logiciels :** 70 % développent des applications web, 52 % des logiciels d'entreprise et 36 % des outils internes.
- **Open source :** 59 % utilisent des outils et des frameworks open source en combinaison avec du logiciel propriétaire.
- **Langages** : Java (47 %), C# (44 %), SQL (41 %) et TypeScript (38 %).

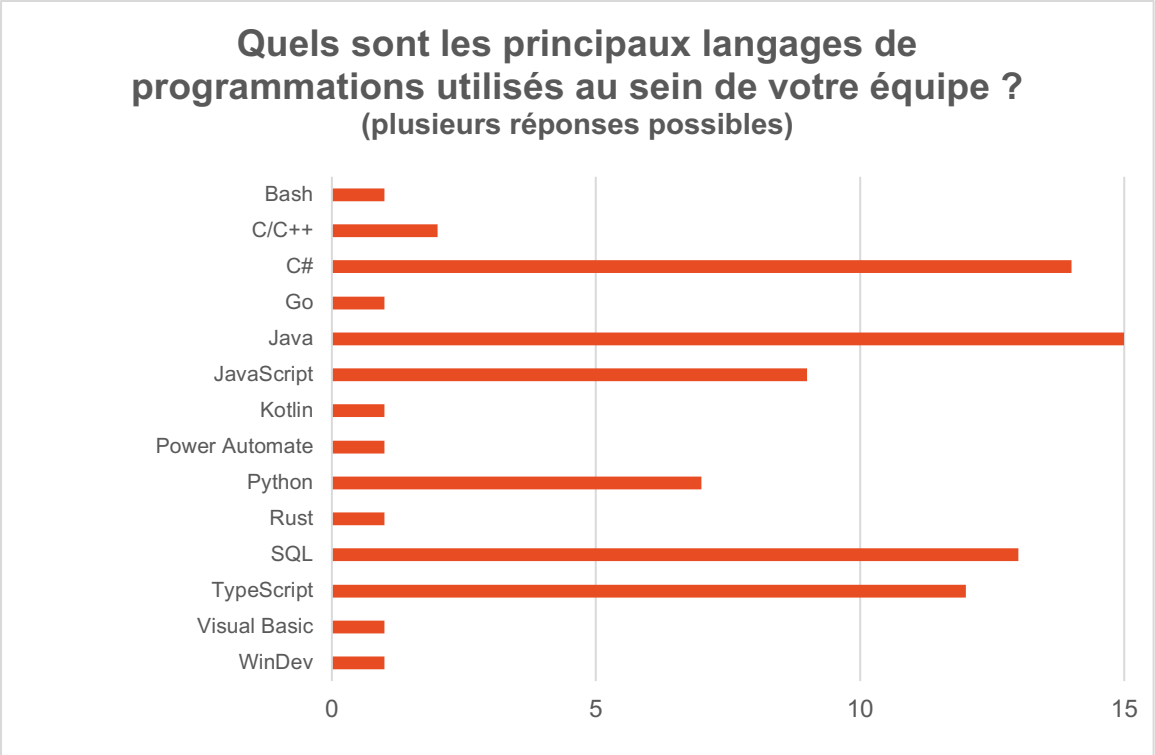
Les **applications web** sont le type de logiciel le plus développé (70 % des répondant·e·s) ; suivi par les **logiciels d'entreprise** tels que les ERPs, CRMs, etc.

(52 %) ; les **outils internes** tels que les frameworks, composants partagés, mécanismes d'intégration, etc. (36 %) ; et les **applications mobiles** (27 %). Parmi les types de logiciels, on retrouve aussi des logiciels scientifiques et techniques de modélisation, de simulation, etc. (24 %), des logiciels système (12 %), de sécurité (12 %), d'IA (12 %) et des logiciels embarqués (6 %). Ces logiciels sont offerts via différents types d'infrastructure : PaaS ou IaaS pour héberger et déployer les produits ; SaaS pour du logiciel accessible via Internet sans installation locale ; et sur site (*on-premise*) pour les services hébergés sur des serveurs en interne. Un·e répondant·e a indiqué que les installations se faisaient directement sur le hardware (sur site).



Une vaste majorité des répondant·e·s indique utiliser des outils et bibliothèques (*frameworks*) **open source** : **principalement** (16 %), **rarement** (22 %) ou de manière **mixte** avec du logiciel propriétaire (59 %). Seul 1 répondant·e indique ne pas du tout utiliser de technologies open source. Parmi les frameworks open source, on retrouve Angular, .Net, Spring, SpringBoot, Hibernate, Django, etc. Parmi les frameworks propriétaires, on retrouve Unity et WinDev.

Enfin, les différents langages de programmation utilisés sont principalement le Java (47 % des répondant·e·s) et le C# (44 %), suivis de SQL (41 %) et TypeScript (38 %).



Au niveau des **environnements de développement** (IDE), 50 % des répondant·e·s indiquent ne pas pouvoir choisir leur IDE, 34 % indiquent devoir choisir dans une liste et seulement 16 % sont libres de choisir leur IDE. La majorité (47 %) utilise Visual Studio Code, suivie d'IntelliJ IDEA (40 %), Visual Studio (23 %), Rider (13 %), Eclipse (7 %), PyCharm (7 %), Vim (7%), CLion (3 %) et Windev (3 %).

Enfin, 72 % des personnes interrogées indiquent utiliser Git comme **système de gestion de versions**, 31 % indiquent utiliser SVN, et seulement 1 répondant·e indique ne pas utiliser de système de gestion de versions.

Expérience développeur

Points clés

- **Satisfaction** : 75 % trouvent leur rythme de livraison tenable, 75 % jugent leur environnement de travail positif.
- **Flexibilité** : 93 % se disent libres ou plutôt libres dans l'organisation de leur travail.

Une majorité de répondant·e·s décrivent l'**expérience globale** au sein de leur équipe comme très positive (46 %) ou plutôt positive (29 %). 11 % la jugent moyenne, manquante de flexibilité et d'innovation, et 11 % la jugent négative, se sentant limitée dans ses choix dans un environnement peu favorable à l'épanouissement. En ce qui concerne la **flexibilité**, 36 % indiquent être très libres d'organiser leurs tâches à leur convenance, 57 % indiquent être plutôt libres, avec des contraintes et des attentes, et seulement 3 % indiquent n'être que moyennement libres, avec peu de contrôle sur

leur emploi du temps et sur la manière de travailler. Enfin, 75 % des répondant·e·s estiment que le **rythme de livraison** du logiciel est tenable, contre 22 % qui affirment le contraire.

En ce qui concerne les **outils de développement**, 50 % des personnes interrogées se disent plutôt satisfaites de ces outils, mais il leur manque quelques fonctionnalités ou possibilités de personnalisation ; 29 % indiquent être très satisfaits ; et 14 % indiquent être moyennement satisfaits, avec des limitations des outils qui ralentissent le travail.

Données démographiques

Âge	Nombre de réponses
Moins de 25 ans	4
25 – 34 ans	11
35 – 44 ans	7
45 – 54 ans	3
55 ans et plus	3
Je préfère ne pas répondre	0

Genre	Nombre de réponses
Femme	1
Homme	26
Non-binaire	1
Autre	0
Je préfère ne pas répondre	0

Expérience en développement logiciel (hors travaux d'études)	Nombre de réponses
Moins de 6 mois	3
6 mois - 1 an	2
1 - 3 ans	4
3 - 5 ans	3
5 - 10 ans	6
Plus de 10 ans	10
Je préfère ne pas répondre	0

Fonction(s) au sein de l' organisation	Nombre de réponses
Architecte logiciel	7
Data scientist	0
Designer UX/UI	2
Développeur backend	18
Développeur frontend	15

Développeur mobile	2
Ingénieur des exigences	1
Ingénieur DevOps	1
Responsable de projet	7
Responsable technique	6
Testeur/Assurance qualité	3
Autre(s)	4

Ancienneté au sein de l'organisation	Nombre de réponses
Moins de 6 mois	3
6 mois - 1 an	2
1 - 3 ans	10
3 - 5 ans	6
Plus de 5 ans	7
Je préfère ne pas répondre	0

Méthodologie

Cette enquête a été conçue dans le cadre du cours « Test et Qualité logicielle » du master en informatique en software engineering à l'Université de Namur, avec pour objectif de dresser un état des lieux des pratiques de qualité logicielle en Fédération Wallonie-Bruxelles. Elle vise à identifier les méthodes, outils et valeurs qui structurent le développement logiciel au sein des organisations locales, afin de rapprocher les contenus pédagogiques des réalités professionnelles. Elle a été réalisée par les étudiants du master sous la supervision d'un chercheur junior et de deux chercheurs senior.

Conception du questionnaire

La conception du questionnaire s'est appuyée sur une analyse de l'état de l'art de la littérature scientifique et de la littérature grise (étude des rapports internationaux tels que *JetBrains Developer Ecosystem*, *State of DevOps Report* et *GitHub Octoverse*).

L'étude et le questionnaire ont été co-créés avec les étudiants lors de sessions de brainstorming, de mindmaps et d'ateliers collaboratifs pour définir les thématiques pertinentes (méthodologies, tests, outils, IA, sécurité, organisation), animées par les deux professeurs en charge du cours. Nous avons suivi un processus itératif, avec plusieurs cycles de relecture et de validation par les enseignants et les étudiants, afin de garantir la clarté et la pertinence des questions.

Le questionnaire final a été structuré en **78 questions**, réparties en grandes catégories : Contexte, Organisation des équipes de développement, Technologies, Automatisation, Sécurité, DevEx, Données démographiques.

L’enquête a été diffusée via les réseaux professionnels et académiques (contacts directs, mailing lists, réseaux sociaux) via la plateforme Drag’n Survey, garantissant l’anonymat des réponses et la conformité au RGPD.

La période de collecte s’est étendue **d’avril à juillet 2025**. Au total, **52 répondants** issus d’organisations de tailles variées (micro-entreprises, PME, ETI, grandes entreprises) ont participé.

Les données ont été agrégées et nettoyées pour éliminer les doublons et incohérences et analysées quantitativement (fréquences, pourcentages) et qualitativement (commentaires libres) pour être visualisées sous forme de graphiques pour faciliter la compréhension des tendances.

Limites de l’étude

Bien que cette enquête fournisse des informations précieuses sur les pratiques de qualité logicielle en Fédération Wallonie-Bruxelles, certaines limites doivent être prises en compte :

Taille et représentativité de l’échantillon

L’étude repose sur 52 répondants, ce qui constitue un échantillon limité par rapport à l’ensemble des organisations actives dans le développement logiciel en FWB. La répartition des répondants n’est pas parfaitement équilibrée : les ETI et PME sont majoritaires, tandis que les micro-entreprises et grandes entreprises sont moins représentées.

Biais de participation

Les répondants sont principalement issus de réseaux académiques et professionnels proches des initiateurs de l’enquête, ce qui peut introduire un biais de sélection. Les organisations ayant des pratiques structurées ou un intérêt marqué pour la qualité logicielle sont probablement surreprésentées.

Auto-déclaration des données

Les réponses reposent sur des déclarations des participants, ce qui peut entraîner des biais liés à la perception individuelle ou à la volonté de présenter des pratiques conformes aux standards. Certaines questions ouvertes ont reçu peu de réponses, limitant la profondeur de l’analyse qualitative.

Temporalité

L’enquête reflète un état des lieux à un instant donné (avril-juillet 2025). Les pratiques évoluant rapidement, notamment avec l’adoption de l’IA et des outils d’automatisation, ces résultats devront être actualisés régulièrement.

Portée géographique

Par choix, l’étude se concentre sur la Fédération Wallonie-Bruxelles. Les conclusions ne sont pas généralisables à d’autres régions ou contextes internationaux.

Conclusion

Cette enquête offre un panorama inédit des pratiques de qualité logicielle au sein de la Fédération Wallonie-Bruxelles. Les résultats mettent en évidence plusieurs tendances fortes :

- **Adoption généralisée de l'Agilité** : Près de 70 % des équipes suivent une approche Agile, bien que son application demeure parfois partielle.
- **Importance croissante des tests et de l'automatisation** : Les tests unitaires et end-to-end sont largement pratiqués, et les pipelines CI/CD contribuent à accroître la rapidité et la fiabilité des déploiements.
- **Usage massif d'outils collaboratifs** : Jira, Confluence et Git dominent, ce qui renforce la traçabilité et la coordination des équipes.
- **Intégration progressive de l'IA** : Plus de 70 % des répondants utilisent des outils propulsés par l'IA, principalement pour la génération de code et l'analyse de la qualité, malgré des défis liés à la formation et à l'interprétabilité.
- **Culture de la collaboration** : Les valeurs de collaboration, de responsabilité et d'autonomie sont largement partagées.

Cette enquête permet également d'identifier plusieurs défis :

- **Manque de formalisation** des guides de développement et des stratégies de sécurité dans certaines organisations.
- **Besoin de formation** pour tirer pleinement parti des outils propulsés par l'IA et des pratiques avancées (CI/CD, analyse statique).
- **Maintien à jour** de la documentation et des dépendances, souvent cité comme un point sensible.
- **Une culture de l'innovation** qui semble faire défaut dans la majorité des équipes de développement.

Cette étude constitue une première étape vers une meilleure compréhension des pratiques locales. Elle ouvre la voie à des initiatives futures : enquêtes régulières, ateliers collaboratifs et renforcement des collaborations entre les mondes académiques et industriels pour soutenir l'amélioration continue de la qualité logicielle en Fédération Wallonie-Bruxelles.