Étude de la qualité architecturale et de l'implémentation des logiciels pour l'IoT

par

Nour KHEZEMI

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 30 AVRIL 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Mme. Naouel Moha, directrice de mémoire
Département de génie logiciel et des TI, École de technologie supérieure

M. Yann-Gaël Guéhéneuc, codirecteur
Département d'informatique et génie logiciel, Concordia

M. Julien Gascon-Samson, président du jury
Département de génie logiciel et des TI, École de technologie supérieure

M. Fabio Petrillo, membre du jury
Département de génie logiciel et des TI, École de technologie supérieure

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

LE 2 AVRIL 2024

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Au terme de ce parcours académique, je tiens à exprimer ma profonde gratitude à toutes les personnes qui m'ont soutenu et accompagné tout au long de cette aventure.

Je tiens tout d'abord à remercier sincèrement mes directeurs de mémoire, Naouel Moha et Yann-Gaël Guéhéneuc, pour leurs précieux conseils, leur encadrement et leur disponibilité tout au long de ce projet. Leur patience et expertise ont été d'une aide inestimable pour donner une forme aboutie à mes idées et m'orienter dans mes recherches.

Je suis également très reconnaissant envers l'ensemble des membres du jury qui ont accepté de consacrer leur temps et leur expertise pour évaluer mon travail. Leurs remarques constructives et leurs encouragements m'ont permis d'enrichir mes réflexions et d'élever la qualité de ce mémoire.

Finalement, je souhaite exprimer ma gratitude à mes proches, ma famille et mes amis, pour leur soutien indéfectible tout au long de mon parcours universitaire. Leur présence et leurs encouragements ont été une source de motivation essentielle.

Étude de la qualité architecturale et de l'implémentation des logiciels pour l'IoT

Nour KHEZEMI

RÉSUMÉ

L'Internet des Objets (IoT) révolutionne la façon dont les gens vivent et travaillent en offrant de nombreux avantages, tels que la capacité d'analyser des données précises en temps réel qui aident les individus à prendre des décisions. Les entreprises peuvent automatiser leurs processus et offrir de nouveaux et meilleurs services avec moins de coûts grâce à l'IoT. Toutefois, pour améliorer la performance et le bon fonctionnement des systèmes IoT, leur qualité doit être étroitement surveillée. Nous considérons la qualité des systèmes IoT directement liée à leur architecture et leur code. Une architecture bien conçue, considérant que nous avons des développeurs compétents, produira un code de qualité. Un code de qualité et une architecture appropriée faciliteront la maintenance et la mise à niveau du système, ce qui contribuera à sa qualité globale.

Dans ce mémoire, nous étudions la qualité architecturale et du code des systèmes IoT.

Choisir un style architectural approprié pour les systèmes IoT nécessite de prendre en compte ses exigences de qualité et de comprendre les compromis entre elles. Les études existantes sur les styles architecturaux IoT n'ont pas identifié les styles architecturaux les plus pertinents lorsque nous prenons en compte les exigences de qualité. Pour cela, nous procédons à une analyse systématique de 103 articles à partir desquels nous étudions tout d'abord les exigences de qualité des systèmes IoT ainsi que les styles architecturaux les plus utilisées dans l'IoT. Nous évaluons ensuite comment chaque style satisfait les exigences de qualité identifiés. Nos résultats peuvent orienter les praticiens des systèmes IoT dans le choix d'un style architectural répondant aux exigences de qualité souhaitées. De plus, nous présentons de nouvelles opportunités de recherche dans les styles architecturaux des systèmes IoT et les exigences de qualité.

Concernant la qualité du code, nous avons noté qu'il existe nombreuses études portant sur la qualité du code des systèmes non IoT, mais il existe un manque de recherche sur la qualité du code des systèmes IoT. En particulier, nous ignorons si le code des systèmes IoT est comparable à celui des non IoT. Sans cette connaissance, nous ne pouvons pas appliquer les résultats et les bonnes pratiques obtenus sur les logiciels non liés à l'IoT en toute confiance aux logiciels IoT.

Par conséquent, nous comparons la qualité du code de deux ensembles équivalents de systèmes IoT et non IoT afin de déterminer s'il existe des similitudes et des différences entre les deux types de logiciels. Nous recueillons également et réexaminons les meilleures pratiques en ingénierie logicielle dans des contextes non IoT afin de les appliquer à l'IoT.

À travers une comparaison des métriques, nous concluons que le code des systèmes IoT est plus complexe, couplé, volumineux, et moins maintenable et cohérent que celui des systèmes non IoT. En tenant compte de ces différences, nous présentons une liste de bonnes pratiques revisitée

avec des approches, des outils ou des techniques pour le développement des systèmes IoT afin de faire face à cet impact.

Sur la base de notre travail, les chercheurs peuvent désormais prendre une décision éclairée quant à l'utilisation des architectures pour répondre aux exigences de qualité, ainsi qu'avoir une idée des différences entre les systèmes IoT et non IoT. Cela leur offre également une perspective sur les bonnes pratiques à adopter lors de développement des systèmes IoT en tenant compte de leurs spécificités.

Mots-clés: IoT, qualité, architecture, implémentation, code

Study of the quality of IoT systems

Nour KHEZEMI

ABSTRACT

The Internet of Things (IoT) is revolutionising the way people live and work by offering numerous benefits, such as the ability to analyze accurate real-time data that helps individuals make decisions. Businesses can automate their processes and provide new and improved services with lower costs through IoT. However, to enhance the performance and proper functioning of IoT systems, their quality must be closely monitored. We consider the quality of IoT systems to be directly linked to their architecture and code. A well-designed architecture, assuming competent developers, will produce quality code. Quality code and appropriate architecture will facilitate system maintenance and upgrades, contributing to its overall quality.

In this thesis, we study the architectural and code quality of IoT systems.

Choosing an appropriate architectural style for IoT systems requires consideration of quality requirements and an understanding of the trade-offs between them. Existing studies on IoT architectural styles have not identified the most relevant styles when considering quality requirements. To address this, we conduct a systematic analysis of 103 articles, in which we first examine the quality requirements of IoT systems and the most commonly used architectural styles in IoT. We then evaluate how each style satisfies the identified quality requirements. Our results can guide IoT system practitioners in choosing an architectural style that meets the desired quality requirements. Additionally, we present new research opportunities in IoT system architectural styles and quality requirements.

Regarding code quality, we observed that numerous studies focus on the code quality of non IoT systems, but there is a lack of research on the code quality of IoT systems. In particular, we do not know if the code for IoT systems is comparable to that of non IoT systems. Without this knowledge, we cannot apply results and best practices obtained from non IoT software to IoT software.

Therefore, we compare the code quality of two equivalent sets of IoT and non IoT systems to determine whether there are similarities and differences between the two types of software. We also collect and revisit software engineering best practices in non IoT contexts to apply them to IoT.

Through a comparison of metrics, we conclude that software for IoT systems is more complex, coupled, larger, and less maintainable and cohesive than non IoT systems. Considering these differences, we present a revisited best practices list with approaches, tools, or techniques for developing IoT systems to address this impact.

Based on our work, researchers can make an informed decision regarding the use of architectures to meet quality requirements and gain insight into the differences between IoT and non IoT systems, thus providing guidance on best practices to use.

Keywords: IoT, Quality, Architecture, Implementation, code

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ETS	École de technologie supérieure
IoT	Internet des Objets
IEEE	Institute of Electrical ana Electronics Engineers
ISO	International Organization for Standardization
SLR	Revues Systématiques de Littérature
SOA	Architecture orientée services
Ioc	Inversion de contrôle
DI	Injection de dépendance
CIC	Cohérence entre les commentaires et les identifiants (CIC)
MDA	Architecture pilotée par modèle
PIM	Modèle indépendant de la plate-forme
PSM	Modèle spécifique à la plate-forme

INTRODUCTION

0.1 Contexte

Les systèmes IoT présentent des avantages significatifs en termes de commodité, d'automatisation et d'efficacité ce qui permet aux individus de mieux vivre et permet aux entreprises d'automatiser les processus et de réduire les coûts de main-d'œuvre. Comme tout logiciel, assurer le bon fonctionnement d'un système IoT nécessite une attention particulière à sa qualité de logiciel. Cette qualité est difficile à garantir considérant qu'elle peut être affectée à différentes étapes du processus de développement.

L'architecture logicielle d'un système IoT doit être conçue pour répondre aux exigences du système tout en maintenant un système de haute qualité. Les chercheurs, développeurs et utilisateurs de l'IoT rencontrent souvent des problèmes avec les systèmes IoT qui résultent de mauvais choix architecturaux. De plus, la qualité du code est un facteur critique dans le succès du développement du système IoT, car elle peut avoir un impact significatif sur la qualité globale du système.

Dans ce mémoire, nous étudions deux axes principaux de qualité des systèmes IoT. En commençant par la granularité la plus large, nous examinons les architectures utilisées avec les systèmes IoT et leurs effets sur les exigences de qualité des systèmes IoT. Nous étudions l'effet des choix architecturaux sur les exigences de qualité des systèmes IoT via une SLR. Ainsi, nous procédons à une étude comparative entre la qualité du code des systèmes IoT et non IoT via des métriques. Nous nous basons sur les différences dans le code entre les systèmes IoT et non IoT démontrées par la comparaison, nous améliorons les bonnes pratiques issues de la littérature pour les systèmes non IoT pour les adapter aux systèmes IoT.

0.2 Problématique

Notre problématique générale est :

Les systèmes IoT présentent des avantages significatifs en termes de commodité, d'automatisation et d'efficacité, mais font également face à des défis liés à leurs parties logicielles. La qualité logicielle des systèmes IoT est donc une préoccupation majeure pour les praticiens et les chercheurs. Les choix architecturaux et la qualité du code développé peuvent influencer cette qualité logicielle d'où la difficulté de la prise de décision éclairée lors du développement de systèmes IoT.

La question centrale de cette recherche réside dans la compréhension approfondie des aspects liés à la qualité architecturale et du code dans les systèmes IoT. Les interrogations principales sont les suivantes :

1. Comment les choix architecturaux impactent-ils les exigences de qualité spécifiques des systèmes IoT ?
2. Quels compromis doivent être considérés lors du choix d'un style architectural, et comment ces choix influencent-ils la qualité du système résultant ?
3. Quelles sont les différences significatives dans la qualité du code entre les systèmes IoT et leurs homologues non IoT ?
4. Comment les bonnes pratiques établies dans le contexte des systèmes non IoT peuvent-elles être adaptées pour répondre aux spécificités des systèmes IoT ?

Notre problématique peut être détaillée en quatre sous problèmes :

Problème 1 :

Nous avons besoin de répertorier toutes les exigences de qualité et les architectures logicielles les plus couramment utilisées dans les systèmes IoT pour comprendre comment les architectures actuelles répondent aux besoins spécifiques de qualité des systèmes IoT. Le grand volume de recherche sur l'architecture IoT peut rendre difficile l'identification des architectures les plus pertinentes et les plus influentes sur la qualité. De plus, aucune étude n'a été réalisée pour vérifier si ces architectures facilitent la mise en œuvre, le contrôle et le respect de ces exigences de qualité.

Problème 2 :

Bien qu'il y ait eu des recherches considérables sur la qualité du code pour le développement de systèmes non IoT, il y a eu peu d'études sur la qualité du code pour les systèmes IoT.

Problème 3 :

Il n'y a aucune étude suggérant que les résultats des études de qualité du code sur les systèmes non IoT sont applicables aux systèmes IoT. Ceci est dû au fait qu'il n'y a pas de comparaison claire entre le code des systèmes IoT et des systèmes non IoT.

Problème 4 :

Nous avons besoin d'orientations pour implémenter les systèmes IoT. Il n'y a pas des travaux exhaustives proposant des bonnes pratiques dans le développement de systèmes IoT de qualité.

0.3 Énoncé du mémoire

Nous pouvons formuler notre énoncé du mémoire comme suit pour résoudre les problèmes présentés précédemment :

Nous étudions les exigences de qualité des systèmes IoT et les architectures les plus utilisées en proposant des recommandations pour la sélection d'architectures répondant aux exigences de qualité. À un niveau de granularité plus fine, nous étudions la qualité du code des systèmes IoT à travers une comparaison et une analyse du code des systèmes IoT à celle des systèmes non IoT. Ces comparaisons orientent nos recommandations des bonnes pratiques pour le développement des systèmes IoT d'où des solutions ciblées pour relever les défis observés.

0.4 Méthodologie du mémoire

Pour répondre aux objectifs de notre mémoire, nous proposons une approche en deux étapes pour chaque aspect de qualité. La figure 0.1 présente un aperçu de notre méthodologie.

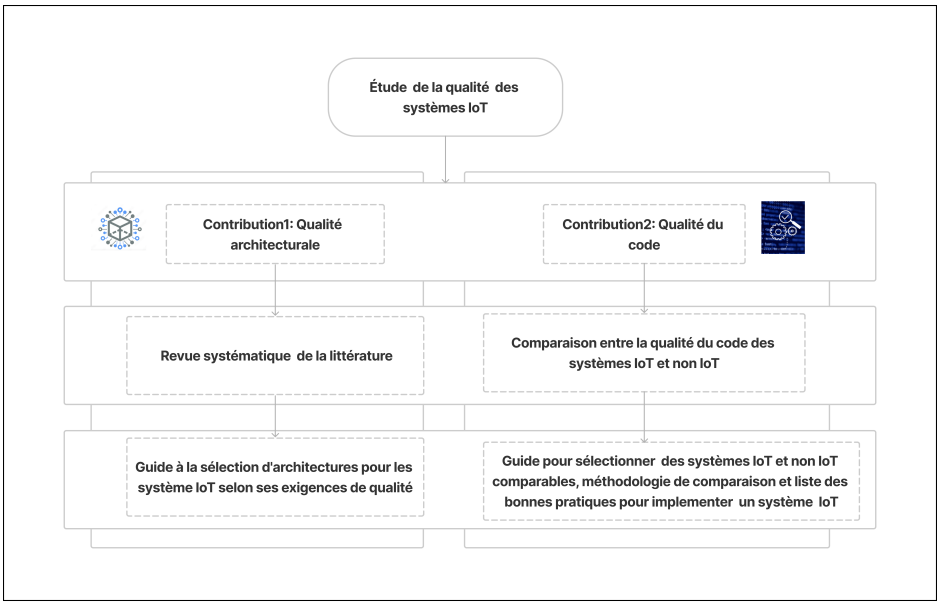

Figure 0.1 Méthodologie du mémoire

Qualité architecturale :

Dans notre quête pour comprendre la qualité architecturale dans les systèmes IoT, nous avons entrepris une démarche méthodologique rigoureuse à travers une SLR. La SLR nous offre une approche exhaustive et systématique pour explorer la vaste étendue de la littérature existante et identifier les informations recherchées en vision d'ensemble de la qualité architecturale. En analysant attentivement 103 études primaires, pour discuter de trois questions de recherche. Ces questions explorent les exigences de qualité pour les systèmes IoT, les architectures logicielles utilisées pour concevoir des systèmes IoT, leurs avantages et inconvénients et si elles répondent aux exigences de qualité des systèmes IoT. En deuxième lieu, cette étape servira comme recommandation à la sélection d'architecture pour les systèmes IoT en considérant les exigences de qualité. Finalement, nous avons identifié des lacunes dans la littérature existante concernant les exigences de qualité et les styles architecturaux, ouvrant la voie à de nouvelles opportunités de recherche.

Qualité du code : Une comparaison de qualité du code des systèmes IoT et non IoT en utilisant des métriques. Le premier résultat de cette étude est la méthodologie permettant de sélectionner

des systèmes IoT et non IoT comparables. Ensuite, la méthodologie de comparaison elle-même constitue un résultat, car d'autres études pourraient s'inspirer de notre approche pour établir leurs propres méthodologies. Enfin, nous proposons une liste des bonnes pratiques pour le développement d'applications IoT de qualité. Ces recommandations s'appuient sur les résultats de notre comparaison ainsi que sur les bonnes pratiques issues des systèmes non IoT.

0.5 Contributions du mémoire

Chapitre 3 : Étude des styles architecturaux des systèmes IoT et de leurs exigences en matière de qualité

La littérature a consacré une attention particulière à l'étude de l'architecture ainsi qu'aux exigences de qualité des systèmes IoT. Cependant, le grand volume de recherche sur l'architecture IoT peut rendre difficile l'identification des architectures les plus pertinentes et les plus influentes sur la qualité. Dans ce chapitre, nous présentons une étude systématique pour examiner et synthétiser systématiquement la littérature sur les exigences de qualité des systèmes IoT et leurs préoccupations architecturales, avec une évaluation de la qualité de ces architectures par rapport aux exigences de qualité. Nous identifions les architectures des systèmes IoT les plus utilisées avec leurs avantages et inconvénients. Nous repérons aussi les exigences de qualité des systèmes IoT pour aider les praticiens à comprendre les critères essentiels pour évaluer ces systèmes au-delà de leurs fonctionnalités. Notre travail aide les utilisateurs IoT à sélectionner une architecture appropriée qui correspond à leurs exigences systèmes et aux fonctionnalités souhaitées. De plus, nos recherches ouvrent des possibilités d'exploration plus approfondie de l'architecture IoT et des exigences de qualité aux communautés de recherche.

En conclusion les principales contributions de cette SLR sont les suivantes :

1. Nous avons compilé des exigences de qualité pour les systèmes IoT afin d'aider les praticiens à comprendre les critères essentiels pour évaluer ces systèmes au-delà de leurs aspects fonctionnels ;
2. Nous avons présenté les styles architecturaux les plus importantes et évaluées comment ils satisfont diverses exigences en matière de qualité. Ces informations visent à informer les praticiens sur la manière dont chaque style architectural répond aux exigences de qualité ;
3. Nous avons fourni des recommandations pratiques aux praticiens en mettant en avant les styles architecturaux qui répondent le mieux à des exigences de qualité spécifiques, pour les aider dans la prise de décision lors du développement de systèmes IoT ;

4. Nous avons formulé des recommandations aux praticiens pour la sélection du style architectural qui correspond le mieux à leurs systèmes, en donnant la priorité aux exigences de qualité attendues ;
5. Nous avons identifié des lacunes dans la littérature existante concernant les exigences de qualité et les styles architecturaux, ouvrant la voie à de nouvelles opportunités de recherche.

Chapitre 4 : Comparaison de la qualité du code et des bonnes pratiques dans les logiciels IoT et non IoT

Notre travail offre des informations sur l'état actuel du développement de l'IoT. C'est une première tentative de comparaison de la qualité du code des systèmes IoT et non IoT. Cette recherche revêt une grande importance pour les développeurs, les chercheurs et les organisations impliquées dans le développement de l'IoT, car elle peut contribuer à améliorer la qualité du code des systèmes IoT.

Notre première contribution est la méthode de sélection de 94 systèmes logiciels équivalents IoT et non IoT à partir de GitHub, garantissant ainsi l'intégrité et la validité de notre analyse comparative, minimisant les biais potentiels de notre recherche. Le processus de sélection veille à ce que les systèmes IoT et non IoT choisis soient comparables en termes de nombre d'étoiles et de forks.

Notre deuxième contribution découle de notre calcul et analyse approfondis de diverses métriques de qualité et systèmes IoT. Les métriques sont essentielles pour notre analyse approfondie, permettant un examen détaillé des résultats basé sur des évaluations méticuleuses du code. Nous fournissons une analyse approfondie d'exemples de systèmes IoT pour illustrer comment nos valeurs de métriques de code se manifestent dans les bases de code des systèmes IoT.

Enfin, nous sélectionnons, discutons et présentons systématiquement une liste revisitée des bonnes pratiques en ingénierie logicielle pour l'IoT, sélectionnée dans la littérature pour chaque catégorie de métriques de code étudiée. Cette liste des bonnes pratiques contient des techniques d'optimisation du code, de la modularité et l'utilisation des patrons de conception, pour résoudre

des défis observés dans la comparaison tels que la complexité élevée, la faible maintenabilité et les problèmes de lisibilité du code des systèmes IoT.

0.5.1 Publications

Ces travaux de recherche ont fait l'objet de deux soumissions dans deux journaux :

1. Khezemi, Minani, Guéhéneuc, Sabir, Moha, El Boussaidi, A Systematic Literature Review of IoT System Architectural Styles and their Quality Requirements, soumis dans IEEE IoT Journal (IoT-J), le 26 février 2024 ;
2. Khezemi, Guéhéneuc, Moha, Comparison of Code Quality and Best Practices in IoT and non-IoT Software, soumis au Elsevier journal of Information and Software Technology (IST), le 27 février 2024.

0.5.2 Paquet de réplication

Nous mettons à disposition tous nos outils, données et résultats d'expérimentation pour nos travaux de recherche sur :

1. Revue systématique de la littérature (https://zenodo.org/records/10547689) ;
2. Comparaison entre les systèmes IoT and IoT (https://zenodo.org/records/10564976).

0.6 Organisation du mémoire

Le reste de ce mémoire est organisé comme suit. Dans le chapitre 1, nous présentons les concepts de base sur lesquels nous appuyons notre recherche. Dans le chapitre 2, nous explorons la littérature existante en examinant les travaux portant sur l'évaluation de la qualité des systèmes IoT, aussi bien au niveau architectural que du code. Dans le chapitre 3, nous présentons notre étude systématique de littérature, dans laquelle nous identifions les exigences de qualité, les architectures IoT couramment utilisées, leurs avantages et leurs inconvénients pour l'IoT et la mesure dans laquelle elles répondent aux exigences de qualité. Dans le chapitre 4, nous proposons une approche pour sélectionner des systèmes IoT et non IoT comparables, ensuite,

nous comparons la qualité du code de ces systèmes à travers des métriques de qualité. Nous mettons en évidence les différences entre les deux types de systèmes et concluons avec des recommandations sur les bonnes pratiques à adopter pour développer des systèmes IoT de bonne qualité. Enfin, nous concluons en soulignant les travaux futurs et en mettant en avant la valeur ajoutée de notre travail dans la résolution de divers problèmes de qualité liés à l'IoT.

CHAPITRE 1

CONCEPTS DE BASE

Ce chapitre explore les notions de base liées aux systèmes IoT et à leurs qualités sur lesquels s'appuie notre travail.

D'abord, nous présentons les définitions de l'IoT, ses domaines d'application, ses avantages et ses défis. Ensuite, nous expliquons la qualité en génie logiciel d'une façon générale, ainsi que les différents niveaux de qualité des systèmes IoT en particulier. En fin de chapitre, nous définissons les termes que nous utilisons tout au long de ce mémoire.

1.1 Internet des Objets (IoT)

L'IoT est un réseau d'objets intelligents qui ont la capacité de partager des informations, des données et des ressources, de réagir et d'agir face à des situations et à des changements dans l'environnement (Madakam, Lake, Lake, Lake *et al.*, 2015). L'IoT est un domaine en constante évolution, cela crée des opportunités pour améliorer l'efficacité, la sécurité et l'expérience utilisateur de diverses applications, mais cela peut également présenter des défis en termes de qualité et de fiabilité (Chen, Xu, Liu, Hu & Wang, 2014).

1.1.1 Domaine d'application IoT

Il y a plusieurs domaines d'application de l'IoT. Nous pouvons citer les domaines suivants comme exemple (Ahmed *et al.*, 2019a) :

1. Les villes intelligentes : L'IoT peut être utilisé pour gérer la circulation, le stationnement, la sécurité publique et la gestion des déchets dans les zones urbaines ;
2. Santé et bien-être : L'IoT peut être utilisé dans diverses applications de soins de santé, telles que la surveillance à distance des patients et la gestion des médicaments ;
3. Agriculture : L'IoT peut être utilisé pour surveiller l'état des sols, la croissance des cultures et les facteurs environnementaux dans l'agriculture, ce qui permet d'améliorer le rendement des cultures et l'efficacité des ressources ;

4. Fabrication et logistique : L'IoT peut être utilisé pour améliorer l'efficacité des processus de fabrication et la gestion de la chaîne d'approvisionnement en assurant une surveillance et un contrôle en temps réel des équipements et des marchandises ;
5. Gestion de l'énergie : L'IoT peut être utilisé pour optimiser la consommation d'énergie et réduire les déchets dans divers contextes, tels que les bâtiments et les systèmes de transport ;
6. Surveillance de l'environnement : L'IoT peut être utilisé pour surveiller des facteurs environnementaux tels que la qualité de l'air et de l'eau, ce qui permet d'améliorer la gestion des ressources et la lutte contre la pollution.

1.1.2 Avantages de l'IoT

L'IoT présente plusieurs avantages, améliorant la vie quotidienne par sa commodité, son automatisation et son efficacité, permettant des décisions informées grâce à des données précises et en temps réel (Driss, Hasan, Boulila & Ahmad, 2021). La communication entre les appareils IoT offre une transparence accrue (Soumyalatha, 2016), favorisant une automatisation étendue et un meilleur contrôle, ce qui se traduit par une production rapide. De plus, l'IoT, en utilisant des capteurs intelligents pour surveiller divers aspects de la vie quotidienne, permet d'économiser du temps et de l'argent (Soumyalatha, 2016). Cela crée de nouvelles opportunités commerciales, stimule la croissance économique et favorise la création d'emplois.

1.1.3 Les défis de l'IoT

La littérature identifie plusieurs défis associés à la mise en œuvre de l'IoT. Ces défis comprennent (Khan, Khan, Zaheer & Khan, 2012) par exemple l'interopérabilité, résultant du développement d'appareils et de systèmes IoT avec des technologies et des normes variées, rend souvent difficile leur communication mutuelle. De plus, la sécurité et la confidentialité sont des préoccupations majeures, car la masse de données générées par les dispositifs IoT peut être vulnérable aux atteintes à la sécurité, entraînant des violations de la confidentialité et des vols de données. La scalabilité constitue un autre défi, avec la croissance rapide du nombre d'appareils IoT, mettant à l'épreuve les infrastructures existantes et nécessitant d'importants investissements dans de

nouvelles infrastructures. L'efficacité énergétique est également une considération cruciale, étant donné que de nombreux dispositifs IoT dépendent de l'énergie de la batterie, limitant ainsi leur utilisation et nécessitant des recharges fréquentes.

1.2 Qualité des systèmes en génie logiciel

La notion de qualité logicielle a été introduite pour la première fois par Weinberg (1971) dans son livre "The Psychology of Computer Programming". Weinberg était un pionnier dans le domaine du développement de logiciels et il a fait valoir que la qualité devait être une préoccupation importante pour les développeurs de logiciels, tout comme elle l'était pour les fabricants de produits manufacturés. Depuis lors, la qualité logicielle est devenue un enjeu crucial pour l'industrie du logiciel, avec des normes et des certifications de qualité développées pour aider à garantir la fiabilité et la performance des logiciels.

1.2.1 Les normes de qualité logicielle

Selon la norme ISO 9000 : 2000, la qualité est définie comme la mesure dans laquelle un ensemble de caractéristiques inhérentes répond aux exigences établies (Arcos-Medina & Mauricio, 2019). En ce qui concerne les logiciels, elle est définie comme la conformité aux exigences fonctionnelles et de performance, aux normes de développement explicitement documentées et aux caractéristiques attendues de tout logiciel développé de manière professionnelle. La production de logiciels de qualité dépend de l'application de normes qui garantissent la satisfaction des exigences fonctionnelles et non fonctionnelles du client.

La norme ISO 9126 de 1991 a été une des principales contributions à la signification des normes de qualité des logiciels. Elle a été remplacée en 2001 par deux normes connexes, l'ISO/CEI 9126 et l'ISO/CEI 14598. La norme ISO/IEC 9126 (ISO, 2003) comprend quatre parties : Le modèle de qualité, Les métriques externes, Les métriques internes et les métriques de qualité en cours d'utilisation. Elle propose un modèle qui classe les attributs de qualité des logiciels en six

caractéristiques (Fonctionnalité, Fiabilité, Convivialité, Efficacité, Maintenabilité et Portabilité), elles-mêmes divisées en sous-caractéristiques.

La norme la plus récente en matière de qualité des logiciels est la norme ISO 25000 :2014 (ISO, 2004), connue sous le nom de SQuaRE (Software Quality Requirements and Evaluation - Exigences et évaluation de la qualité des logiciels). Elle a été fondée sur les principes énoncés dans les normes ISO 9126 et ISO 14598 et est divisée en cinq thèmes : Gestion de la qualité, Modèle de qualité, Mesures de qualité, Exigences de qualité et Évaluation de la qualité.

1.2.2 Différence entre système IoT et non IoT

Le terme IoT, désigne le réseau collectif d'appareils connectés et la technologie qui facilite la communication entre les appareils et le cloud, ainsi qu'entre les appareils eux-mêmes (Amazon, 2023). Un système IoT se caractérise par l'utilisation d'un réseau d'objets physiques interconnectés (tels que des capteurs, des actionneurs et des microcontrôleurs et des processeurs embarqués), capables d'échanger des informations entre eux et de réagir aux changements de leur environnement. Ces objets sont dotés de communication, permettant la connectivité entre les humains, les objets et les interactions entre les objets eux-mêmes. Cette connectivité offre une compréhension de l'environnement et permet des réponses rapides aux situations complexes, caractérisant ainsi l'essence de l'IoT. Comme exemple, nous pouvons citer les Voitures connectées et les villes intelligentes.

En revanche, les systèmes non IoT se réfèrent à des environnements ou des systèmes informatiques qui ne sont pas intégrés dans le réseau global de l'IoT et ne possèdent pas les fonctionnalités de communication et de réactivité caractéristiques des systèmes IoT (Madakam *et al.*, 2015). Parmi les exemples de systèmes non IoT, nous pouvons citer les applications de bureau ou mobiles traditionnelles, les systèmes logiciels d'entreprise et même certains systèmes embarqués qui ne sont pas connectés à l'Internet.

En raison de ces différences, les mesures logicielles existantes peuvent ne pas être adéquates pour évaluer la qualité des systèmes IoT et doivent donc être révisées pour relever ces défis.

1.3 Qualité des systèmes IoT

Un nombre de chercheurs, d'ingénieurs et d'experts en IoT ont contribué à développer des normes et des méthodologies pour évaluer la qualité des systèmes IoT.

1.3.1 Les normes de qualité des systèmes IoT

L'IEEE (Institute of Electrical ana Electronics Engineers) a créé un groupe de travail dédié à la qualité des systèmes IoT (IEEE P1931) qui a publié une norme intitulée "Standard for Architectural Framework for the Internet of Things (IoT) Systems Quality of Service (QoS)" en 2019. Cette norme fournit un cadre pour évaluer la qualité de service des systèmes IoT.

De même, l'ISO (International Organization for Standardization ; Organisation internationale de normalisation) a publié une série de normes pour les systèmes IoT, y compris des normes relatives à la sécurité, à la gestion des données et à la qualité de service. La norme ISO/IEC 30141, publiée en 2018, fournit des lignes directrices pour l'ingénierie de la qualité des systèmes IoT. La qualité peut être définie comme la capacité d'un système IoT à répondre aux exigences et aux attentes de ses utilisateurs en termes de fonctionnalités, de performance, de sécurité et de convivialité. Pour assurer une qualité élevée, il est nécessaire de comprendre les différents facteurs qui peuvent affecter la qualité d'un système IoT, tels que les architectures adoptées, les technologies utilisées, les environnements de déploiement, les interactions entre les différents composants et les processus de développement. Dans ce qui suit, nous présentons les différents aspects de qualité à étudier dans le cas des systèmes IoT dans notre recherche.

1.3.2 Qualité d'architecture IoT

L'architecture d'un système IoT fait référence à la conception du système, y compris ses composants logiciels, son infrastructure réseau et ses protocoles de communication. La qualité de l'architecture des systèmes IoT, peut être définie a travers l'évaluation des exigences de qualité, telles que la scalabilité, fiabilité, la sécurité, l'évolutivité, la maintenabilité, la performance et d'autres aspects non fonctionnels.

1.3.2.1 Style architectural, architecture et patron de conception

Le terme style architectural comporte les termes architecture et patron, qui se rapportent tous deux à des solutions récurrentes visant à résoudre des défis de conception ou architecturale pour différents types d'applications logicielles. Cependant, une distinction fondamentale existe entre les deux : le patron est l'abstraction d'une forme existante qui se répète dans des contextes spécifiques et non arbitraires (Appleton, 1997). En revanche, l'architecture est la structure des relations entre différents composants d'un système (Garlan & Perry, 1995; Washizaki *et al.*, 2020). Pour la suite de cette étude, en se basant sur le travail de Muccini, Spalazzese, Moghaddam & Sharaf (2018), nous utiliserons systématiquement les termes architecture pour nommer les architectures en couches, SOA, l'architecture basée sur le cloud, microservices, architecture REST et architecture orientée évènement en tant qu'architecture. En ce référent au travail de Tekinerdogan & Köksal (2018), Pair à Pair, Filtres et Tubes, Client Serveur et Publish Subscribe sont des patrons de conception.

1.3.3 Qualité du code des systèmes IoT

La qualité du code source peut être évaluée par les mesures des métriques. Ces métriques sont divisées en catégories telles que la taille, la redondance, la complexité, le couplage, la couverture des tests unitaires, la cohésion, la lisibilité du code, la sécurité et l'hétérogénéité du code. Aussi, pour mener une évaluation approfondie, des modèles de qualité robustes tenant compte de diverses dimensions et facteurs peuvent être utilisés.

Notre mémoire se concentre sur l'étude de la qualité de l'architecture et du code des systèmes IoT.

1.4 Conclusion

Dans ce chapitre, nous avons en premier lieu présenter les concepts fondamentaux des systèmes IoT ; la définition de l'IoT, ses domaines d'application ainsi que ses avantages et défis. Ensuite,

nous avons souligné la différence entre les systèmes IoT et non IoT, abordé la qualité en génie logiciel et exploré les différents niveaux de qualité spécifiques aux systèmes IoT.

L'étude approfondie de la qualité de l'architecture et du code des systèmes IoT constituera le cœur de notre mémoire. En combinant ces connaissances, nous sommes bien équipés pour entreprendre une analyse approfondie des systèmes IoT et fournir des recommandations pratiques pour améliorer leur qualité.

Le prochain chapitre se concentrera sur la revue de la littérature, mettant en lumière les travaux connexes à notre recherche. Nous justifierons l'importance de notre travail et expliquerons pourquoi il est nécessaire de le mener à bien.

CHAPITRE 2

REVUE DE LITTÉRATURE

Dans ce chapitre, nous présentons les études liées à notre sujet de recherche sur la qualité des systèmes IoT.

Tout d'abord, nous présentons les travaux portant sur la qualité des systèmes IoT les plus pertinents ce qui nous permet de situer notre recherche dans le contexte plus large des travaux existants. Ensuite, nous examinons les travaux qui se concentrent sur les architectures utilisées en IoT ainsi que ceux portant sur les exigences de qualité des systèmes IoT. Ensuite, nous présentons les recherches portant sur la qualité du code des systèmes IoT. Nous abordons également les travaux sur la comparaison entre les systèmes IoT et non IoT. Enfin, nous concluons avec un résumé de ces travaux et une discussion des leurs limitations, qui justifient le besoin de notre travail de recherche.

2.1 Qualité des systèmes IoT

Dans ce mémoire, nous étudions deux axes principaux de qualité des systèmes IoT. En commençant par la granularité la plus large, nous examinons les architectures utilisées avec les systèmes IoT et leurs effets sur les exigences de qualité des systèmes IoT. Ensuite, nous étudions la qualité du code des systèmes IoT. Comme illustré dans la figure 2.1, chaque catégorie de qualité a ses exigences de qualité.

Une architecture bien conçue permettra de produire une conception technique solide, qui permettra de produire du code de qualité qui à son tour permettra de répondre aux exigences du système tout en maintenant un système de haute qualité.

2.1.1 Qualité des architectures des systèmes IoT

La littérature a montré un intérêt considérable pour l'étude des architectures et des exigences de qualité des systèmes IoT. Cependant, le grand nombre de recherches sur l'architecture IoT peut

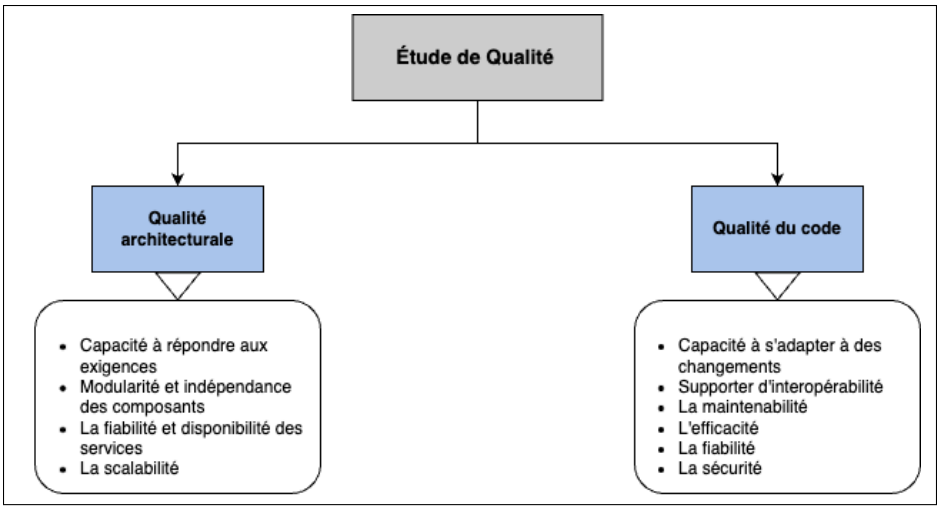

Figure 2.1 Les types d'études de qualité

rendre complexe l'identification des architectures les plus pertinentes et influentes en termes de qualité.

Le tableau 2.1 présente une brève comparaison entre les travaux que nous avons répertoriés portant sur les architectures IoT et exigences de qualité et notre contribution.

Comme présenté dans le tableau 2.1, les travaux comme ceux de Stojanov & Dobrilovic (2021); Sethi & Sarangi (2017); Gill, Behbood, Ramadan-Jradi & Beydoun (2017) offrent une vue partielle des architectures IoT, décrivant leurs composants, leurs modèles de communication et leurs domaines d'application, tout en soulignant les défis et exigences inhérents à ces systèmes. Certains travaux, tels que ceux de Muccini & Moghaddam (2018b); Stefanova-Stoyanova, Stoyanov & Danov (2021); Washizaki *et al.* (2020), ont proposé différents styles architecturaux et patrons de conception pour le développement d'IoT. Ces approches mettent l'accent sur des composants décentralisés et autonomes favorisant leur indépendance fonctionnelle. De plus, ils offrent une vue d'ensemble complète des modèles adaptés aux systèmes IoT, incluant la communication, la sécurité et la gestion des données. D'autres articles comme (Zhen, Zeng, Chen, Li & Liu, 2011; Fahmideh & Zowghi, 2018; Stojanov & Dobrilovic, 2021) se sont penchés sur des architectures spécifiques pour des types particuliers de systèmes IoT. Par exemple, Fahmideh & Zowghi (2018) a évalué diverses architectures IoT pour les villes intelligentes,

discutant de leurs avantages, inconvénients et de leur applicabilité face à des défis spécifiques, tandis que le travail de Stojanov & Dobrilovic (2021) s'est concentré sur les exigences de qualité pour l'architecture logicielle des systèmes IoT basés sur des capteurs en couches.

Notre travail se distingue des travaux présenter au-dessus en considérant une liste plus large d'architectures IoT à étudier avec leurs avantages et inconvénients, tout en détaillant les exigences de qualité des systèmes IoT et en les mettant en relation avec chaque architecture présentée. En outre, nous mettons l'accent sur les architectures les plus citées, explorant leur compatibilité avec les exigences de qualité d'IoT. Notre travail couvre davantage ces aspects que les travaux existants, d'où la nécessité de notre contribution.

2.1.2 Qualité du code des systèmes IoT

Klima *et al.* (2022) ont résumé des métriques de qualité de code pertinentes des systèmes IoT et évalué leur impact sur la qualité générale des systèmes selon les normes ISO/IEC.

Ils ont classé ces métriques en fonction de la taille (nombre de lignes de code), de la complexité (complexité cyclomatique), du couplage (Response For Class), etc. Ces métriques offrent une évaluation précise de la qualité du code des systèmes IoT, et nous les utiliserons et les présenterons dans notre comparaison, nous permettant d'évaluer systématiquement et de juxtaposer la qualité du code entre ces systèmes logiciels IoT et non IoT.

Bien que notre étude adopte une approche similaire en utilisant ces métriques établies de qualité de code des systèmes IoT, notre travail s'étend au-delà de l'évaluation des métriques. Nous entreprenons une analyse comparative approfondie entre les systèmes IoT et non IoT, utilisant ces métriques pour explorer les différences ces deux types de systèmes.

Corno, De Russis & Sáenz (2020) ont étudié le développement de logiciels open source dans les systèmes IoT et non IoT, analysant 60 projets. Ils ont trouvé des différences significatives dans les processus de développement, la spécialisation des développeurs et la réutilisabilité du code entre ces deux types de systèmes. Leur étude a également examiné les contributions

Tableau 2.1 Comparaison des études connexes sur les architectures IoT

Étude	Objectif	Type	Année	Portée	Concentration		
					EQ	SA	CAEQ
Sethi & Sarangi (2017)	Propose un aperçu des architectures et technologies IoT	Enquête	2017	N.A	◴	◴	◴
Gill *et al.* (2017)	Identifie les défis architecturaux de l'IoT et les solutions pertinentes	SLR	2017	23 papiers de 2014 à 2017	○	◐	○
Muccini & Moghaddam (2018a)	Adresse une classe des styles architecturaux et des modèles IoT, et fournit une cartographie durable à utiliser comme cadre pour apprendre et évaluer les styles, modèles et descriptions architecturaux	Étude de cartographie systématique	2018	63 articles jusqu'en 2018	○	◕	○
Fahmideh & Zowghi (2018)	Analyse comparative de neuf architectures IoT existantes bien connues	Comparaison	2018	N.A	○	◕	○
Muccini *et al.* (2018)	Analyse un ensemble de modèles de distribution et d'auto-adaptation IoT pour identifier leurs combinaisons architecturales appropriées	Analyse	2018	N.A	◐	◐	◐
Alshohoumi, Sarrab, AlHamadani & Al-Abri (2019)	Évolution de l'architecture IoT	SRL	2019	2008-2018	◴	◴	○
Washizaki *et al.* (2020)	Architectures et concepts pour l'IoT, ainsi que l'indication des directions d'amélioration lors de l'adoption de ces architectures IoT	SLR	2020	2014-2018	●	◕	○
Razzaq (2020)	Démontre les styles architecturaux, modèles et logiciels pour construire des logiciels IoT	Analyse	2020	140 articles de 2005 à 2020	○	◐	○
Stojanov & Dobrilovic (2021)	Présente une expérience subjective avec des attributs de qualité tels que la scalabilité, la maintenabilité, la sécurité, la disponibilité et la portabilité lors de la conception d'un système IoT basé sur des capteurs en couches	Expérience	2021	N.A	○	◕	○
Stefanova-Stoyanova *et al.* (2021)	Discute des principaux modèles d'architectures logicielles les plus utilisés, les mieux documentés et les plus étudiés	Analyse	2021	N.A	○	●	○
Alfonso, Garcés, Castro & Cabot (2021)	Les compromis des architectures auto-adaptatives dans les systèmes IoT.	SLR	2021	N.A	○	◴	○
Siddiqui, Khendek & Toeroe (2023)	Architectures basées sur les Microservices pour les systèmes IoT.	SLR	2023	N.A	○	◕	○
Márquez, Astudillo & Kazman (2023)	Tactiques architecturales	SMS	2023	N.A	○	◐	○
Our study	Offre des perspectives sur les exigences de qualité et les styles architecturaux associés aux systèmes IoT.	SLR	2024	2012-2023	●	●	●

Acronyms : EQ - Exigences de qualité, SA - Style architecturale, CAEQ - Correspondance Architecture/Exigences de Qualité, NA - Non Disponible

des développeurs, les modifications de fichiers, la spécialisation et la maturité des projets en analysant les dépendances des projets.

Notre étude mesure des métriques de code pour comparer la qualité du code des systèmes IoT et non IoT. Bien que l'approche de Corno *et al.* (2020) diffère en termes d'objectifs de recherche et de méthodes, ces études se complètent mutuellement pour comprendre les systèmes IoT.

Alors que la littérature suggérait la complexité des systèmes IoT (Corno *et al.*, 2020; Gubbi, Buyya, Marusic & Palaniswami, 2013; Taivalsaari & Mikkonen, 2017; Larrucea, Combelles, Favaro & Taneja, 2017), dans ce travail nous réalisons une comparaison quantitative pour évaluer cette complexité par rapport aux systèmes non IoT.

Larrucea *et al.* (2017) ont souligné le manque de pratiques d'ingénierie logicielle établies pour les systèmes IoT et ont souligné le besoin d'une orientation efficace dans l'ingénierie des systèmes IoT. Pour cela, nous sélectionnons, étudions et fournissons une liste des bonnes pratiques pour le développement de systèmes IoT tirées de la littérature.

À notre connaissance, aucun travail antérieur n'a comparé la qualité du code des systèmes IoT et non IoT. Notre étude vise à identifier les similitudes ou les différences entre les systèmes IoT et non IoT en analysant des métriques logicielles spécifiques, offrant une approche simple et efficace pour sélectionner des systèmes comparables. Notre travail ouvre la voie à l'étude de la qualité des systèmes IoT, faisant progresser la compréhension et l'amélioration des pratiques de développement IoT.

2.2 Synthèse générale

Les études examinées dans ce chapitre démontrent l'importance d'étudier la qualité des systèmes IoT. Dans notre recherche, nous choisissions d'étudier les aspects architecturaux et de code car nous considérons ces deux aspects constituent des piliers fondamentaux pour garantir la qualité. Les travaux proposent différentes approches et mesures pour améliorer leur qualité. Ils donnent un aperçu des différentes exigences de qualité importantes pour ces systèmes, comme les défis

de sécurité, de complexité et d'interopérabilité. De plus, cela nous donne une idée des travaux existants. Le grand nombre de recherches sur l'architecture IoT rend difficile l'identification des architectures les plus pertinentes et les plus influentes par rapport aux exigences de qualité existantes. Par notre contribution, nous visons à synthétiser la littérature sur les exigences de qualité des systèmes IoT, sur les architectures à leur disposition, sur leurs fonctionnalités et si ces architectures garantissent les exigences de qualité du système IoT trouvées. Aussi, il n'y a également pas de directives de bonnes pratiques pour le développement de systèmes IoT. En plus, il y a peu des travaux existants sur la qualité du code des systèmes IoT et il n'y a pas un travail qui compare le code des systèmes IoT et non IoT. Cela limite notre compréhension de la manière dont les systèmes IoT diffèrent des systèmes non IoT en termes d'exigences de qualité et de défis. Il se peut que nous perdions également l'opportunité d'utiliser des modèles de qualité existants pour les systèmes non IoT s'ils sont similaires sur certains points, car il y a une grande maturité dans la recherche sur la qualité des systèmes non IoT.

Pour remédier à ces limites, notre travail se concentrera sur les exigences de qualité d'évaluation des systèmes IoT, puis sur les architectures existantes pour ces systèmes. De plus, des études empiriques supplémentaires qui comparent les systèmes IoT avec les systèmes non IoT peuvent fournir des idées précieuses sur les défis et les opportunités uniques des systèmes IoT. Ainsi, nous présentons des recommandations visant à améliorer la qualité d'architecture des systèmes IoT et le développement du code.

CHAPITRE 3

ÉTUDE DES STYLES ARCHITECTURAUX DES SYSTÈMES IOT ET DE LEURS EXIGENCES EN MATIÈRE DE QUALITÉ

Comme abordé dans le chapitre 2, de nombreux travaux ont été réalisés dans la littérature sur les styles architecturaux des systèmes IoT, mettant en évidence la nécessité d'une synthèse regroupant l'ensemble de ces styles architecturaux. De plus, il n'existe pas de liste complète et définitive des exigences de qualité associées aux styles architecturaux existants pour les systèmes IoT.

Dans ce chapitre, nous entreprenons une Revue Systématique de la Littérature (Systematic Literature Review (SLR)) sur les travaux traitant des styles architecturaux et les exigences de qualité des systèmes IoT.

Bien que certaines SLR aient été menées ces dernières années, se concentrant sur les styles architecturaux des systèmes IoT, comme présenté dans la sous-section 2.1.1, aucune n'a abordé simultanément les styles architecturaux et les exigences de qualité. Ainsi, nous réalisons une SLR pour examiner ces aspects, évaluant comment chaque style architectural répond à ces exigences de qualité.

Nos Questions de Recherche (QRs) sont les suivantes :

1. **QR1** : Quelles sont les exigences de qualité pour les systèmes IoT ?
2. **QR2** : Quels sont les styles architecturaux des systèmes IoT et leurs avantages et inconvénients ?
3. **QR3** : Les styles architecturaux identifiés dans la QR2 répondent-ils aux exigences de qualité énoncées dans la QR1 ?

Pour répondre à ces QRs, nous suivons les directives du PRISMA pour les revues systématiques (Page *et al.*, 2021). Nous examinons 20 616 études potentiellement pertinentes issues de huit bibliothèques numériques, publiées jusqu'en 2023. En appliquant des critères d'inclusion et d'exclusion, nous procédons à une sélection rigoureuse. L'évaluation de la qualité des études s'est effectuée sous différentes perspectives, incluant la conception, la conduite, l'analyse, les

conclusions et les implications. Au terme de ce processus, nous retenons 103 études primaires (PSs). L'analyse approfondie de ces PSs nous a permis de présenter les résultats relatifs aux styles architecturaux IoT, aux exigences de qualité, et à la manière dont les styles architecturaux identifiés abordent différentes exigences de qualité.

Les résultats de ce chapitre peuvent servir comme recommandation pour à la fois les praticiens et les chercheurs en IoT en :

1. Choisissant des styles architecturaux appropriés qui s'alignent sur les exigences de qualité souhaitées, améliorant la prise de décision dans le développement de systèmes IoT ; et
2. Identifiant de nouvelles opportunités de recherche dans les styles architecturaux et les exigences de qualité des systèmes IoT, présentant des voies pour une exploration et une avancée ultérieures dans le domaine.

Dans la suite de ce chapitre, nous examinons la méthodologie de recherche adoptée, détaillée dans la section 3.1. Ensuite, nous présentons les résultats obtenus en réponse à nos questions de recherche dans la section 3.2, suivis d'une discussion des résultats et de leurs implications dans la section 3.3. La section 3.4 aborde les potentiels biais pouvant influencer la validité des résultats. Enfin, nous concluons ce chapitre en proposant quelques perspectives d'avenir.

3.1 Méthodologie

Nous suivons les directives PRISMA pour mener une revue systématique de la littérature (SLR) afin de répondre à nos questions de recherche (QRs) (Page *et al.*, 2021). Nous effectuons trois activités principales : planifier, examiner et rendre compte de nos conclusions. Au cours de la phase de planification, nous définissons l'objectif de cette étude, tel que décrit dans la section précédente. Nous définissons notre principale question de recherche comme :

Dans quelle mesure les différents styles architecturaux IoT satisfont-elles à des exigences de qualité spécifiques ?

Le processus de révision pour mener cette SLR est résumé dans la figure 3.1. Nous définissons cinq étapes : 1) définir les questions de recherche, 2) formuler la stratégie de recherche, 3) identifier les études, 4) faire le snowballing et 5) extraire et analyser les données.

Le étapes des directives PRISMA est illustré dans la figure 3.2 avec les nombres des articles résultants de chaque étape.

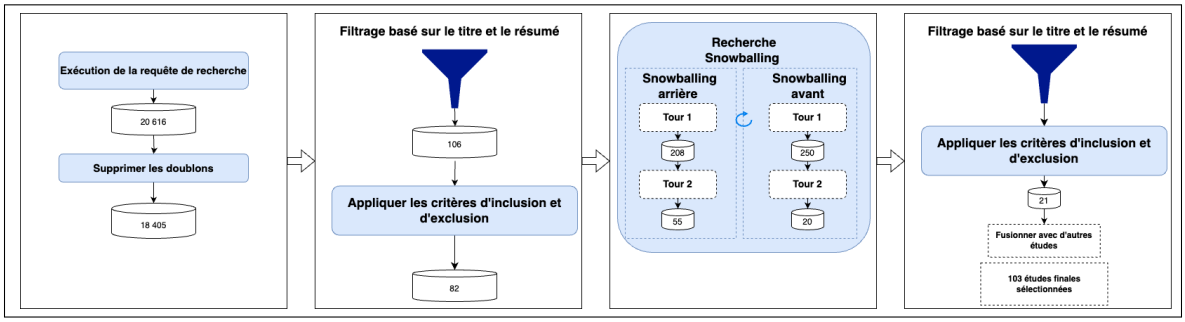

Figure 3.1 Méthodologie de recherche de la SLR

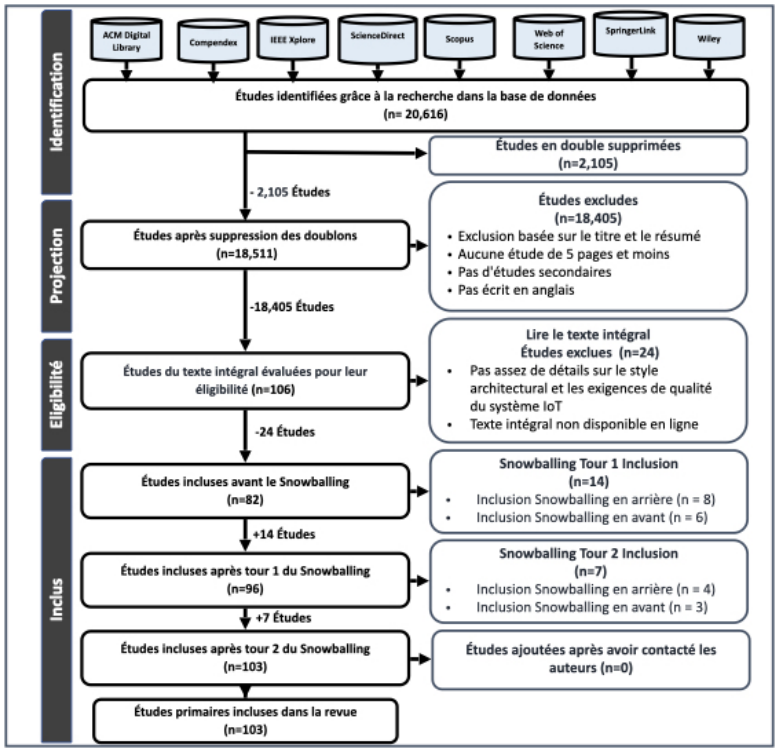

Figure 3.2 Processus PRISMA

3.1.1 Questions de recherche

Ce chapitre répond aux QRs suivants :

1. **QR1** : Quelles sont les exigences de qualité pour les systèmes IoT ?

   Justification : Cette QR vise à identifier et répertorier les exigences de qualité essentielles pour les systèmes IoT. Comprendre ces exigences est crucial pour une évaluation approfondie de la qualité des systèmes IoT.
2. **QR2 :** Quels sont les styles architecturaux des systèmes IoT et leurs avantages et inconvénients ?

   Justification : Cette QR se concentre sur les styles architecturaux utilisés dans les systèmes IoT, examinant leurs caractéristiques distinctes, leurs avantages et leurs inconvénients. En comprenant de manière exhaustive les options architecturales disponibles et en les évaluant, les practiciens peuvent prendre des décisions éclairées. L'identification des avantages et des inconvénients associés à chaque architecture nous aide à mieux comprendre leur applicabilité et leurs limites pour répondre à des exigences de qualité spécifiques.
3. **QR3 :** Les styles architecturaux identifiés dans la QR2 répondent-ils aux exigences de qualité énoncées dans la QR1 ?

   Justification : Cette question vise à relier les styles architecturaux de la QR2 aux exigences de qualité de la QR1, évaluant comment ces styles architecturaux répondent à ces exigences de qualité. Cette évaluation aide à identifier les styles architecturaux qui répondent le mieux à des exigences de qualité spécifiques, aidant ainsi les praticiens à choisir l'architecture la plus adaptée à leur cas d'utilisation.

3.1.2 Stratégie de recherche

Nous avons formulé notre stratégie de recherche en utilisant le cadre PICO (Population, Intervention, Comparison, Outcome) (Cooke, Smith & Booth, 2012). Nous avons repéré PICO comme suit :

1. Population : Études englobant les systèmes IoT et leurs styles architecturaux ;
2. Intervention : Discussions, évaluations ou analyses liées à l'architecture IoT ;

3. Comparaison : Comparaisons des avantages, inconvénients et évaluations des styles architecturaux IoT ;
4. Résultat : Identification des exigences de qualité des systèmes IoT, puis des styles architecturaux IoT les plus largement utilisées.

Pour appliquer PICO, nous avons suivi les étapes suivantes :

1. Extraction des termes clés de la question principale de recherche ;
2. Identification des synonymes possibles des termes principaux ;
3. Application du OU (OR) logique pour combiner les synonymes possibles des termes principaux ;
4. Application du ET (AND) logique pour combiner les expressions des étapes précédentes.

En conséquence, nous formulons la requête de recherche suivante :

((IoT) OR (internet of things) OR (internet-of-things)) AND ((Software architecture) OR (architecture) OR (architecting) OR (architectural) OR (design)OR (designing) AND ((disadvantages) OR (inconvenient) OR (drawbacks) OR (negative side) OR (problems)) OR ((advantages) OR (benefits) OR (added value))) AND ((IoT) AND ((evaluation) OR (evaluating) OR (quality) OR (quality attributes) OR (requirements) OR (quality characteristics)))

L'utilisation du cadre PICO (Population, Intervention, Comparison, Outcome) permet de structurer la requête de recherche de manière à cibler spécifiquement les études pertinentes pour répondre aux questions de recherche. En identifiant les termes clés associés à chaque composante du cadre PICO et en combinant ces termes de manière logique, la requête est conçue pour être exhaustive tout en ciblant précisément les aspects pertinents pour les objectifs de l'étude. Cette approche permet d'être sûr que les articles importants n'ont pas été manqués en fournissant une méthodologie systématique pour la recherche et la sélection des études. En utilisant le cadre PICO, nous sommes plus convaincus que nous avons couvert toutes les bases nécessaires pour identifier les études pertinentes dans le domaine de l'architecture IoT et de ses exigences de qualité. La requête de recherche formulée en utilisant le cadre PRISMA joue un

rôle crucial dans l'exhaustivité et précision des résultats obtenus en plus de la méthodologie de recherche détaillée qui garantit une couverture large des articles étudiés. Cependant, il est important de reconnaître les menaces à la validité associées à cette approche de recherche. Par exemple, la sélection des termes clés et des synonymes peut être subjective et influencer les résultats de la recherche. De plus, l'exhaustivité de la recherche dépend de la pertinence des termes sélectionnés et de leur utilisation appropriée dans la requête. En utilisant les cadres PICO et PRISMA, cette méthodologie de recherche met en évidence la rigueur méthodologique de l'étude, ce qui renforce la confiance dans les résultats obtenus.

3.1.3 Identification des études

3.1.3.1 Identification des bibliothèques numériques

Nous sélectionnons 8 bibliothèques numériques : ACM Digital Library, Compendex, IEEE Xplorer, ScienceDirect, Scopus, SpringerLink, Web of Science et Wiley. Ces bibliothèques numériques sont sélectionnées en suivant les bonnes pratiques énoncées dans le travail de Dyba, Dingsoyr & Hanssen (2007). Des bibliothèques numériques telles que Web of Science (WoS), Scopus, IEEE Xplorer et ScienceDirect sont choisies pour leur couverture étendue dans diverses disciplines scientifiques, assurant un large spectre d'études liées à l'IoT. Pour des contenus plus spécialisés concernant les domaines de l'informatique et de l'ingénierie, ACM Digital Library et Compendex sont sélectionnées pour leurs ressources sur mesure, offrant une couverture complète des styles architecturaux IoT et des sujets connexes. D'autres bibliothèques numériques telles que Wiley et SpringerLink sont incluses en raison de leur réputation bien établie dans la publication d'articles universitaires, notamment ceux centrés sur les systèmes et styles architecturaux IoT. Certaines bibliothèques numériques ont des limitations de requête (par exemple, ScienceDirect permet huit connecteurs, SpringerLink interdit les parenthèses et la bibliothèque numérique ACM rejette les caractères génériques). Nous adaptons notre stratégie de recherche pour chaque bibliothèque numérique. Nous avons récupéré 20 616 études après l'exécution de la requête de recherche dans chaque bibliothèque numérique, comme indiqué dans la Figure 3.2.

3.1.3.2 Suppression des études en double

Nous supprimons 2 105 études en double, en comparant les titres et les auteurs des études, et conservons 18 511 études.

3.1.3.3 Sélection

Critères d'inclusion et d'exclusion

Les critères d'inclusion et d'exclusion sont établis sur la base de lignes directrices spécifiques.

1. **Critères d'inclusion :**
    a. L'étude se concentre sur les styles architecturaux IoT et les exigences de qualité des systèmes IoT ;
    b. L'étude est publiée dans une conférence, un journal, un atelier ou un chapitre de livre ;
    c. L'étude est rédigée en anglais ;
    d. L'étude est primaire ;
    e. L'étude comporte au moins 5 pages.
2. **Critères d'exclusion :**
    a. L'étude n'est pas rédigée en anglais ;
    b. L'étude n'est pas disponible en ligne ;
    c. L'étude comporte moins de 5 pages (y compris les références) ;
    d. L'étude ne fournit pas suffisamment de détails sur le style architectural des systèmes IoT et les exigences de qualité.

Pendant ce processus, nous appliquons nos critères d'inclusion et d'exclusion pour identifier les PSs. Nous appliquons les trois premières critères d'inclusion et avons exclu 18 405 études. Nous conservons 106 études après le premier tri.

Nous appliquons les deux derniers critères d'exclusion pour exclure 24 études et retenons 82 études.

3.1.4 Snowballing

Les résultats de ce processus sont résumés dans le Tableau 3.1.

Snowballing tour 1

Nous examinons les références de toutes les PSs pour identifier des études supplémentaires pertinentes. Nous recherchons toutes les études référencées par ces 82 études. Nous éliminons toute étude récupérée lors de notre recherche initiale et des études ne se concentrant pas sur le style architectural IoT ou les exigences de qualité, et nous retenons 208 études potentiellement pertinentes. Nous passons en revue de ces 208 études potentiellement pertinentes en appliquant les mêmes critères d'inclusion et d'exclusion qu'auparavant et avons trouvé 5 PSs supplémentaires. De même, nous utilisons Google Scholar pour identifier toutes les études pertinentes citant les 82 études sélectionnées. Nous éliminons toute étude récupérée précédemment et des études ne se concentrant pas sur le style architectural IoT ou les exigences de qualité. Nous retenons 250 études potentiellement pertinentes pour un examen approfondi. Nous passons en revue de ces 250 études potentiellement pertinentes en supprimant les doublons et en appliquant les mêmes critères d'inclusion et d'exclusion qu'auparavant. Nous ajoutons 9 PSs. À la fin de cette première étape, nous trouvons 14 PSs supplémentaires, portant notre total à 98 PSs.

Snowballing tour 2

Nous examinons les 14 études trouvées dans le cycle précédent et avons recommencé le processus de snowballing pour elles. Nous vérifions toutes les références utilisées dans ces 14 études et avons trouvé 75 nouvelles études potentielles. Nous passons en revue ces 75 études sur la base des critères d'exclusion et d'inclusion définis précédemment. Après cette étape, nous obtenons 7 PSs supplémentaires, portant le nombre total d'études considérées dans cette revue à 103 PSs.

Le nombre de PS pour chaque bibliothèque numérique est présenté dans la Figure 3.3.

Tableau 3.1 Snowballing en avant et en arrière

Snowballing	Cycle	Récupéré	Inclus
Arrière	Tour 1	208	5
Avant	Tour 1	250	9
Arrière	Tour 2	55	4
Avant	Tour 2	20	3
Total		**533**	**21**

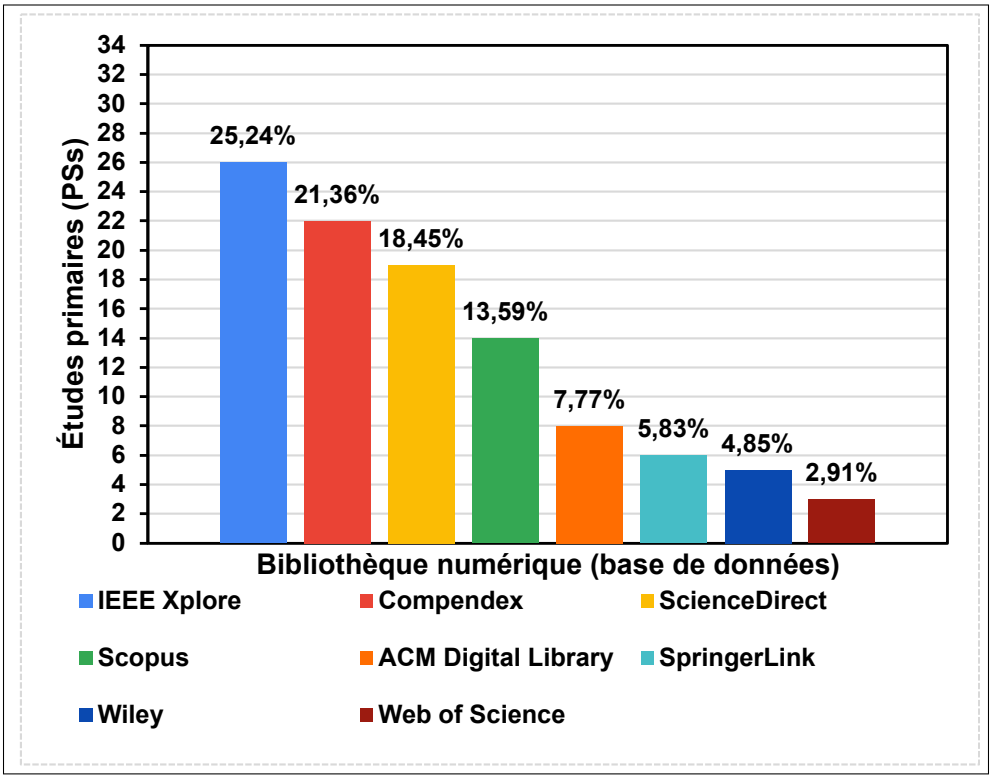

Figure 3.3 PSs de chaque bibliothèque numérique

IEEE Xplore arrive en tête avec 26 PSs (25.24%), suivi de Compendex avec 22 PSs (21.36%). ScienceDirect se classe ensuite avec 19 PSs (18.45%), suivi de Scopus avec 14 PSs (13.59%). La bibliothèque numérique ACM en compte 8 (7.77%), tandis que SpringerLink en a 6 PSs (5.83%). Wiley et Web of Science en ont chacun 5 (4.85%) et 3 (2.91%) PSs respectivement.

3.1.5 Extraction et analyse des données

Pour obtenir les données nécessaires pour répondre à chaque question de recherche (QR1-QR3), nous étudions chaque PS. Nous extrayons les différents éléments de données décrits dans le Tableau 3.2. Le formulaire d'extraction de données que nous avons utilisé est disponible publiquement sur le site de Zenodo (https://zenodo.org/records/10547689), garantissant la transparence et facilitant les efforts potentiels de réplication.

Tableau 3.2 Éléments d'extraction de données

#	Élément de données	Description	QRs
1	Code	Identifiant unique du PS.	
2	Titre	Le titre du PS.	
3	Auteur	Les auteurs du PS.	
4	Année de Publication	Année de publication du PS.	
5	Source	La source du PS.	
6	Type	Le type du PS.	
7	Résumé	Le résumé du PS.	
8	Domaine Cible	Le domaine cible du PS.	
9	Exigences de Qualité (QRs)	QRs présentées dans le PS.	QR1
10	QRs de l'Architecture	QRs satisfaites par l'architecture.	[QR1-QR3]
11	Style d'Architecture	Les styles architecturaux trouvés dans le PS.	QR2
12	Langage de Développement (DL)	Le DL présenté dans le PS.	
13	Expérience et Résultats	Résultats présentés dans le PS.	
14	Domaine d'Application	Domaine d'application IoT.	
15	Principaux Défis	Principaux défis présentés dans le PS.	

Pendant ce processus nous examinons indépendamment 103 PSs. Le texte est lu, tous les PSs précédemment identifiés sont soumis à un examen approfondi pour garantir leur alignement avec nos objectifs de recherche. Malgré les critères de sélection stricts appliqués précédemment, cette étape vise à identifier d'éventuels cas atypiques ou des articles qui pourraient être exclus. Cependant, aucun article non pertinent n'est identifié au cours de cette phase. Par conséquent, tous les PSs ayant passé les phases de sélection précédentes sont considérés comme cohérents avec notre focus de recherche, et donc aucune exclusion supplémentaire n'est faite à ce stade.

Nous extrayons ensuite de chaque PS les informations significatives qui semblent pertinentes pour les questions de recherche de notre SLR. Pendant le processus d'extraction des données,

nous suivons deux étapes. Initialement, nous extrayons les données, puis nous les vérifions. Cette phase de vérification implique une vérification des informations extraites avec les PSs originaux, facilitant les discussions et résolvant tout écart ou divergence dans l'interprétation ou l'extraction.

3.2 Résultats

Cette section détaille les résultats obtenus pour répondre à nos questions de recherche.

Dans les sous-sections suivantes, nous décrivons les résultats de chaque QR. Nous commençons d'abord par les exigences de qualité discutées pour les systèmes IoT (QR1). Ensuite, nous discutons des styles architecturaux des systèmes IoT abordés dans les PSs (QR2). Enfin, nous évaluons comment les styles architecturaux identifiés répondent aux exigences de qualité discutées (QR3).

3.2.1 QR1 : Quelles sont les exigences de qualité pour les systèmes IoT ?

Nous utilisons le terme exigences de qualité des systèmes IoT pour décrire le processus d'analyse de la qualité d'un système IoT. En identifiant et extrayant ces exigences de qualité des systèmes IoT des PSs étudiés, nous avons pu éclairer et répondre aux questions de recherche, en particulier à la QR1, permettant ainsi une analyse approfondie des critères déterminants pour évaluer la qualité des systèmes IoT. Cela permet de comprendre comment garantir la qualité d'un système IoT.

Le tableau 3.3 présente la liste des exigences de qualité identifiées dans les PSs. Dans les paragraphes suivants, nous fournissons la définition et des explications de ces exigences de qualité, examinant leur pertinence dans le contexte de l'IoT, et décrivant comment elles sont appliquées pour améliorer la qualité des systèmes IoT.

Sécurité émerge comme l'exigence de qualité la plus fréquemment discutée dans 20 PSs. Les menaces de sécurité pour les systèmes IoT augmentent à mesure que davantage de dispositifs sont

Tableau 3.3 Exigences de qualité des systèmes IoT

Exigences de qualité des systèmes IoT	**#PSs**
Sécurité	20
Scalabilité	14
Performance	11
Disponibilité	10
Interopérabilité	8
Fiabilité	8
Confidentialité	8
Gestion de l'énergie	6
Flexibilité	4
Évolutivité	3

connectés aux réseaux. Nous avons trouvé que les systèmes IoT qui utilisent des clés de sécurité composites pour chiffrer et déchiffrer les données dans le réseau IoT contribuent à satisfaire cette exigence (Sun & Bai, 2021). Les caractéristiques de sécurité peuvent jouer un rôle significatif dans le bon fonctionnement d'un système IoT particulier (Liao, Nazir, Khan & Shafiq, 2021), telles que la portée maximale du signal, la variété des topologies réseau, la sécurité du transfert de données, l'intégrité, la tolérance aux pannes, etc. Il est nécessaire d'établir une méthode commune pour évaluer les différents niveaux de sécurité pour différents types de données entrantes à différents niveaux de granularité. Ces méthodes communes d'évaluation peuvent améliorer le service de sécurité pour différents systèmes IoT (Sicari, Cappiello, De Pellegrini, Miorandi & Coen-Porisini, 2016).

Scalabilité émerge comme l'exigence de qualité suivante la plus fréquemment discutée avec 14 PSs. Scalabilité se réfère à la capacité du système à utiliser, ajouter ou réduire des ressources comme des dispositifs, de la mémoire, du CPU, etc., pour traiter les demandes variables de manière optimale (Ismail, Hamza & Kotb, 2018). Le système peut étendre sa capacité et ses limites chaque fois que nécessaire sans affecter l'architecture de base sous-jacente et ses capacités (Jacob & Mani, 2018). Évaluer la scalabilité de l'ensemble du système IoT est difficile. Cela dépend de la facilité d'ajout de ressources supplémentaires en cas de besoin. La scalabilité du

système IoT peut être évaluée selon trois dimensions : la capacité de traitement est améliorée en augmentant la capacité de traitement des composants logiciels, allant de l'utilisateur au serveur. La capacité d'information est élargie en augmentant la capacité de stockage dans la base de données du système et dans le stockage local des composants logiciels individuels. La connectivité peut être améliorée en augmentant le nombre de points d'accès pour les utilisateurs et les éléments de détection (Nicholson, Chawathe, Chen, Noble & Wetherall, 2006).

Performance est la prochaine exigence de qualité discutée avec 11 PSs. Performance fait référence à la capacité des systèmes IoT à traiter de grandes quantités de données, à les traiter rapidement et à fournir des réponses en temps opportun aux utilisateurs finaux. L'exigence de performance est cruciale dans un système IoT, car le système souvent gère une énorme quantité de données générées par des dispositifs connectés. Elle peut être affectée par divers facteurs tels que la connectivité réseau, la capacité de traitement des données, la capacité de stockage et la capacité du système à s'adapter aux changements de l'environnement (Ali, Ali & Badawy, 2015). Certains chercheurs ont proposé des critères pour mesurer la performance à travers des métriques dans leurs expériences, telles que le délai de bout en bout, le taux de perte de paquets, le débit, la consommation d'énergie (Said & Tolba, 2016; Ibrahim *et al.*, 2019), le transfert et le nombre de messages transmis lors de l'émulation du système IoT (Rao, Schelén & Lindgren, 2016). Le concept de performance englobe la performance du réseau et des protocoles de communication (Akram *et al.*, 2021).

Disponibilité est une autre exigence de qualité populaire discutée avec 10 PSs. Disponibilité garantit que le système IoT peut accéder aux données, aux services et aux stockages quel que soit l'état de l'infrastructure sous-jacente et soit accessible pour les usagers. Un système IoT doté d'une solution de stockage redondante garantit que le système est disponible en permanence. La disponibilité d'un système garantit que les composants essentiels du système facilitent un accès ininterrompu et en temps réel aux services pour les utilisateurs finaux (Krylovskiy, Jahn & Patti, 2015). Cette exigence de qualité est associée à la tolérance aux pannes et à la réduction des temps d'arrêt pour les réparations du système ou des composants spécifiques (Stojanov & Dobrilovic, 2021).

Interopérabilité est discutée dans 8 PSs. Interopérabilité est la capacité des différents dispositifs et systèmes IoT à travailler ensemble (Kugler, Czwick, Nguyen & Anderl, 2021). L'interopérabilité favorise l'interaction, l'ajout et le déploiement d'objets hétérogènes à travers les couches architecturales. L'architecture d'un système IoT doit fournir des mécanismes, des protocoles et des techniques permettant l'interopérabilité, tels que le bus de services d'entreprise (ESB), l'orchestration de processus ou de workflows, les adaptateurs et les enveloppes (Fahmideh & Zowghi, 2018).

Fiabilité est discutée dans 8 PSs. La fiabilité présente la capacité du système à fonctionner de manière constante et à fournir de manière précise et exacte les fonctionnalités attendues par les utilisateurs finaux (Jacob & Mani, 2018; Ibrahim *et al.*, 2019). Compte tenu de la nature d'inter-traitement et d'intercommunication des dispositifs impliqués (Jacob & Mani, 2018), afin de garantir sa qualité.

Confidentialité est discutée dans 8 PSs. La confidentialité porte sur la régulation de l'accès aux données par les utilisateurs individuels et cherche à garantir que les données collectées sont transmises exclusivement à des entités autorisées (Jacob & Mani, 2018). Le déploiement de technologies de communication sans fil dans les systèmes IoT introduit de nouvelles préoccupations en matière de confidentialité, notamment des vulnérabilités liées à l'accès à distance. Le développement de mécanismes efficaces de préservation de la confidentialité pour l'IoT reste une tâche ardue. Cette exigence de qualité nécessite la création d'un modèle complet capable de représenter les entités et les relations de l'IoT, ainsi que la mise en œuvre de mécanismes dynamiques de contrôle d'accès aux flux de données pour répondre à l'échelle et à la nature toujours changeante des scénarios IoT (Miorandi, Sicari, De Pellegrini & Chlamtac, 2012).

Gestion de l'énergie est discutée dans 6 PSs. La gestion de l'énergie implique que les dispositifs IoT fonctionnent de manière cohérente pendant de longues périodes sans nécessiter un remplacement fréquent des piles. Un système IoT de haute qualité fournit des mécanismes pour les capteurs, les actionneurs et les serveurs qui minimisent la consommation d'énergie

(Miorandi *et al.*, 2012; Fahmideh & Zowghi, 2018). Un exemple concret serait l'utilisation de capteurs intelligents qui s'activent uniquement lorsqu'un mouvement est détecté, réduisant ainsi la consommation d'énergie. Cette optimisation énergétique est liée à la qualité du système IoT en permettant une gestion efficiente de l'énergie, l'un des aspects essentiels pour garantir sa performance globale.

Flexibilité est discutée dans 4 PSs. Elle fait référence à la capacité de s'adapter ou de s'ajuster aux changements de circonstances ou d'exigences (Kyriazopoulou, 2015). Cela signifie également la capacité d'incorporer un nouveau composant sans perturber le service aux utilisateurs (Thomas, Onyimbo & Logeswaran, 2016). La flexibilité permet une plus grande agilité et une meilleure réactivité aux conditions environnementales changeantes.

Évolutivité est l'exigence de qualité la moins discutée avec 3 PSs. Elle fait référence à la capacité d'un système à évoluer et à s'adapter avec le temps. Les systèmes IoT sont soumis à des changements continus, comme les avancées technologiques, les besoins commerciaux changeants et les exigences évolutives des utilisateurs. L'évolutivité permet aux systèmes de rester efficaces et précieux avec le temps (Krylovskiy *et al.*, 2015), même en cas d'évolution de la technologie et des besoins des utilisateurs. Lors de l'évaluation de l'évolutivité d'un système IoT, nous considérons trois types de modules principaux : les dispositifs connectés, les plates-formes externes et les applications métier internes, comme décrit par Vermeire, Faes, De Bruyn & Verelst (2019). La capacité d'évolvabilité dépend de l'ouverture du système au changement et à l'extension, comme le soulignent Muccini *et al.* (2018).

Les exigences de qualité telles que l'hétérogénéité, la mobilité, la résilience, la récupérabilité, la découverte des ressources, la configuration des objets, l'accumulation des données, l'analyse des données, la visualisation des données, la facilité de mise en œuvre, la maturité, la modularité, la documentation, l'apprentissage et la compréhension, sont moins fréquemment abordées dans les PSs, chaque exigence étant citée une seule fois. En raison de leur popularité limitée, nous avons délibérément choisi de ne pas les aborder de manière approfondie dans cette étude.

Recommandation pour les praticiens : La sécurité, la scalabilité, la performance, la disponibilité, la fiabilité, l'interopérabilité, la confidentialité, la gestion de l'énergie, la flexibilité grâce à une conception modulaire, et l'évolutivité pour une efficacité à long terme sont quelques-unes des nombreuses exigences de qualité qui doivent être prises en compte lors de la mise en œuvre de projets de systèmes IoT. Les développeurs doivent prioriser les exigences de qualité en fonction des aspects critiques du cas d'utilisation du système IoT identifié. Les praticiens doivent analyser les besoins spécifiques de leur système et définir quelles exigences doivent être prises en compte. Ils doivent également rester informés sur d'autres exigences de qualité qui ne sont pas encore discutées ou auxquelles on accorde moins d'importance.

Recommandation pour les chercheurs : À travers l'analyse des exigences de qualité discutées, nous pouvons mettre en évidence de nombreux autres aspects, tels que la connectivité, la distributivité, la dynamique et l'utilisation des ressources, qui peuvent avoir été négligés. Nous suggérons la nécessité de mener davantage d'études pour explorer toutes les exigences de qualité possibles jugées importantes pour tout système IoT en indiquant celles qui sont nécessaires et celles qui sont facultatives pour tout système IoT.

3.2.2 QR2 : Quels sont les styles architecturaux des systèmes IoT et leurs avantages et inconvénients ?

Le choix de l'architecture implique de prendre en compte des facteurs tels que l'hétérogénéité, divers domaines d'application, des fonctionnalités variées et une large gamme de composants (Waris, Jaleel, Shoaib, Nigar & Abalo, 2022).

De plus, les systèmes IoT ont des applications et des exigences diverses, ce qui conduit à différents choix des styles architecturaux. Les chercheurs ont proposé diverses architectures et patrons de conception (Burhan, Rehman, Khan & Kim, 2018). Nous présentons les styles architecturaux trouvés dans les PSs étudiés dans le Tableau 3.4.

SOA est une architecture destinée à la construction de logiciels d'entreprise composée des services. C'est l'architecture la plus fréquemment discutée avec 22 PSs. Chaque service est

Tableau 3.4 Styles architecturaux pour les systèmes IoT

Architecture	**# PSs**
SOA	22
L'architecture en couches	21
Microservices	15
Architecture basée sur le Cloud	6
Publish Subscribe	5
Pair à Pair	5
Client Serveur	4
Tubes et Filtres	4
Architecture orientée événements	4
Architecture REST	2

responsable et détient une seule fonction ou une partie d'une fonction (Magaia *et al.*, 2020). L'architecture orientée services (SOA) est un choix architectural viable pour les systèmes IoT.

SOA est fréquemment utilisée et intégrée dans la plupart des dispositifs du monde réel qui facilitent le traitement et la communication des tâches utilisateur (Ahmed *et al.*, 2019a). SOA peut également faciliter le traitement de différentes tâches utilisateur sur plusieurs dispositifs utilisés par différents types de systèmes IoT (Ahmed *et al.*, 2019a). Chaque dispositif peut être conçu pour fournir un service ou une fonction spécifique pouvant être utilisé par d'autres dispositifs ou services dans l'architecture. Cette architecture facilite également le contrôle de chaque dispositif IoT de manière indépendante (Ahmed *et al.*, 2019a).

Cependant, SOA présente certains inconvénients, notamment la sous-utilisation des ressources système, la dépendance à une seule technologie, la réduction du coût de développement et de l'efficacité opérationnelle, ainsi que le risque d'enflure du code, comme le soulignent Sun, Liang & Huang (2020). Le développement de dispositifs IoT a également rencontré des défis liés à l'interopérabilité, pouvant avoir des répercussions sur divers intervenants, comme l'ont observé Butzin, Golatowski & Timmermann (2016).

Les systèmes IoT opèrent dans un environnement dynamique, où les entités peuvent rejoindre ou quitter le réseau à tout moment. Cependant, l'architecture SOA n'est pas intrinsèquement adaptée pour prendre en compte les caractéristiques dynamiques des systèmes IoT, ce qui pose des défis pour la conception des services dans le cadre de l'architecture SOA.

Architecture en couches émerge comme l'architecture suivante la plus discutée avec 21 PSs. Elle permet aux développeurs d'organiser les composants ayant des fonctionnalités similaires en couches horizontales. Chaque couche remplit un rôle spécifique au sein de l'application (Stefanova-Stoyanova *et al.*, 2021). Les systèmes IoT sont généralement structurés à l'aide d'une architecture en trois couches, c'est-à-dire la couche réseau, la couche de perception et la couche d'application (Burhan *et al.*, 2018; Gavrilović & Mishra, 2021; Sithole & Marshall, 2020; Javadpour, Wang & Rezaei, 2020). La couche de perception est responsable de la collecte initiale des données provenant des objets connectés dans le système à l'aide de technologies telles que les étiquettes d'identification par radiofréquence (RFID), les capteurs et les caméras. La couche réseau transmet les données collectées depuis la couche de perception et gère l'infrastructure matérielle et logicielle. La couche d'application englobe toutes les interactions humaines au sein du système.

Cependant, les chercheurs ont proposé une architecture en cinq couches pour l'IoT en ajoutant une couche métier et une couche d'application (Khan *et al.*, 2012; Jia, Komeily, Wang & Srinivasan, 2019; Javadpour *et al.*, 2020). L'ajout de ces couches comprend la couche métier, qui se concentre sur la gestion et la facturation des systèmes IoT tout en protégeant la vie privée des utilisateurs. La couche d'application contrôle les systèmes IoT et favorise le développement de nouvelles applications pour améliorer l'intelligence, l'authentification et la sécurité au sein de la plate-forme (Khan *et al.*, 2012; Jia *et al.*, 2019; Javadpour *et al.*, 2020).

Des modèles comportant 7 à 9 couches ont également été élaborés pour offrir une structuration plus élaborée et détaillée des différents niveaux fonctionnels. Ces modèles approfondis visent à segmenter davantage les étapes de traitement des données et de communication au sein des systèmes IoT. Ces modèles plus étendus s'avèrent bénéfiques pour des applications exigeantes

nécessitant une maîtrise précise et détaillée des interactions et des processus au sein des systèmes IoT.

L'un des avantages de l'architecture en couches est sa modularité, car elle permet des changements dans chaque couche tout en maintenant des interfaces cohérentes (Stefanova-Stoyanova *et al.*, 2021; Kyriazopoulou, 2015). Cette approche architecturale offre également une assurance d'utilisabilité en facilitant la réutilisation cohérente de chaque couche et améliore la testabilité (Stefanova-Stoyanova *et al.*, 2021). À mesure que l'écosystème IoT continue d'évoluer, la complexité croissante des appareils et la richesse des données associées posent de nouveaux défis en matière de traitement des données. En réponse à ces défis, une architecture en trois couches, composée d'appareils, de passerelles intelligentes et de centres de données, peut être utile.

Cependant, l'architecture en couches peut présenter certains inconvénients dans le contexte de l'IoT. Un tel inconvénient est la réduction potentielle de la productivité due à l'introduction de multiples couches, comme l'a souligné Stefanova-Stoyanova *et al.* (2021). De plus, la fiabilité devient un problème lors de l'utilisation de l'architecture en couches, car une défaillance dans une couche peut rendre l'ensemble du système inopérable. Le débogage des systèmes IoT utilisant une architecture en couches peut être complexe, car il peut être difficile de localiser exactement la source des problèmes, comme l'a noté Ibrahim *et al.* (2019).

Architecture microservices est l'architecture suivante discutée avec 15 PSs. Elle est une évolution de l'architecture SOA (Muccini & Moghaddam, 2018a). Elle organise une application en un groupe de services faiblement couplés qui communiquent à l'aide de protocoles légers. L'objectif principal est que les équipes puissent développer et déployer ces services de manière indépendante les uns des autres (Sun *et al.*, 2020).

L'architecture microservices offre des avantages tels que ceux discutés par Santana, Alencar & Prazeres (2018) : l'hétérogénéité technologique, la résilience, la facilité de déploiement (du Plessis, Mendes & Correia, 2021), l'alignement organisationnel et l'agilité (Ahmed *et al.*, 2019a), qui permettent le développement de systèmes IoT à grande échelle. Tout comme SOA,

chaque service ou microservice est encapsulé pour garantir la possibilité de modification, ce qui facilite la réutilisation.

Un autre avantage partagé par l'architecture SOA et l'architecture microservices, particulièrement pertinent dans le contexte de l'IoT, est leur capacité à faciliter le développement de services modulaires faiblement couplés qui s'intègrent de manière transparente dans divers systèmes. Cette caractéristique est cruciale dans les systèmes IoT, où différents appareils et composants doivent souvent travailler ensemble. La flexibilité offerte par ces approches architecturales garantit que les modifications apportées à un service ou un microservice n'ont pas d'effet en cascade sur l'ensemble du système, permettant une adaptabilité et une évolutivité plus fluides.

Les inconvénients de l'architecture SOA et de l'architecture microservices sont liés aux performances et aux protocoles de communication. Compte tenu de la nature complexe et largement distribuée des systèmes IoT, la mise en œuvre de protocoles de communication efficaces entre différents composants devient un défi supplémentaire pour ces architectures, comme l'indique une enquête menée par Kumar, Mouli & Kumar (2017).

De plus, la division du système en instances de services ou de microservices conteneurisés entraîne une utilisation accrue de la mémoire. Pour certains appareils embarqués, il est difficile d'augmenter la consommation de mémoire en raison de divers facteurs tels que les contraintes matérielles et l'espace physique limité disponible pour l'expansion de la mémoire. Il est nécessaire de prêter une attention particulière à la sécurité, à la confidentialité et à la gouvernance lors de l'utilisation de l'architecture SOA et des microservices (Dragoni *et al.*, 2017). De plus, la conception et la mise en œuvre de ces architectures peuvent s'avérer complexes, en particulier pour les systèmes à grande échelle (Sun *et al.*, 2020; Stefanova-Stoyanova *et al.*, 2021).

Architecture basée sur le Cloud est une autre architecture populaire avec 6 PSs. Elle utilise le cloud computing comme partie centrale de son architecture informatique (Muccini & Moghaddam, 2018a). Elle comprend trois composants : le cloud IoT, les dispositifs IoT et la console de gestion de l'utilisateur, qui est généralement une application mobile pour contrôler les dispositifs (Jin *et al.*, 2022).

Les solutions basées sur le cloud peuvent gérer des tâches complexes liées au stockage, au traitement des données en temps réel et à l'optimisation du volume important de données dans un système IoT (Meloni, Pegoraro, Atzori, Benigni & Sulis, 2018).

Les styles architecturaux basés sur le Cloud peuvent facilement s'adapter à la demande de l'application, ce qui est particulièrement important dans les systèmes IoT, où de grandes quantités de données sont générées et traitées en temps réel. De plus, Pena, Sarkar & Maheshwari (2015); Sethi & Sarangi (2017) ont discuté que l'architecture basée sur le Cloud est conçue pour être scalable, flexible et adaptable à différents types de systèmes IoT et de cas d'utilisation.

Cependant, comme inconvénients, les styles architecturaux basés sur le Cloud nécessitent un accès internet fiable et rapide pour fonctionner correctement. Dans les régions où certaines installations IoT peuvent être déployées, la connectivité internet médiocre ou les interruptions fréquentes pourraient entraîner une diminution des performances et de la disponibilité.

Publish Subscribe (Publier Abonner) est discutée dans 5 PSs. Ce patron architectural composé principalement de deux composants, à savoir les Éditeurs (Publishers) et les Abonnés (Subscribers) (Tekinerdogan & Köksal, 2018). Il s'agit d'un style de messagerie dans lequel les éditeurs (Publishers) catégorisent les messages en classes sans avoir connaissance des abonnés spécifiques (Subscribers), par rapport à ces messages qui sont directement envoyés aux abonnés individuels (Banijamali, Pakanen, Kuvaja & Oivo, 2020).

L'architecture Publish Subscribe est facilement modifiable car les éditeurs et les abonnés sont faiblement connectés. Il peut être utilisé comme une approche de coordination qui contribue à améliorer le fonctionnement de différentes parties du système IoT. Il réduit le couplage entre les composants et améliore les performances du système (Barroca Filho & de Aquino Junior, 2018; Tekinerdogan & Köksal, 2018).

Comme inconvénient, les performances de l'architecture Publish Subscribe peuvent être affectées négativement par les services événements agissant comme des intermédiaires entre les éditeurs et les abonnés, facilitant la distribution et le routage des événements, car ils ajoutent de la

complexité de traitement et augmentent la latence du système (Jacob & Mani, 2018). De plus, le débogage et la gestion des erreurs lors de l'utilisation de cette architecture sont complexes. La disponibilité est également un problème dans cette architecture, car les services d'événements sont un point potentiel de défaillance (Stefanova-Stoyanova *et al.*, 2021).

Client Serveur est discutée dans 5 PSs. C'est une architecture typique pour les applications Web. Il s'agit d'une structure d'application distribuée qui répartit les tâches entre les serveurs, qui fournissent des ressources ou des services, et les clients, qui demandent ces services (Banijamali *et al.*, 2020). Le serveur dans l'IoT peut être un dispositif de stockage central, de traitement ou de surveillance, tandis que les clients peuvent être des dispositifs mobiles (Jacob & Mani, 2018).

Cette architecture facilite la modification des services communs, car ils n'ont besoin d'être modifiés qu'en un seul endroit. Elle permet également un contrôle centralisé sur les ressources, qui peuvent être réparties entre plusieurs emplacements dans les systèmes IoT.

L'architecture Client Serveur permet d'augmenter la puissance de calcul des appareils finaux, tels que les ordinateurs, les téléphones portables, les routeurs, etc. Elle permet aussi d'améliorer la capacité du matériel englobant tout ce qui est physique comme les processeurs, les cartes graphiques, les mémoires, commutateurs, les routeurs et du réseau (Banijamali *et al.*, 2020). Cette amélioration du réseau peut être réalisé à travers l'ajout des ressources supplémentaires à un serveur existant pour augmenter sa capacité. Ainsi que l'utilisation de technologies telles que les SDN (Software-Defined Networking), la virtualisation réseau, le multiplexage, peut aider à augmenter la capacité du réseau en offrant des fonctionnalités avancées de gestion du trafic.

Cependant, comme inconvénient, le serveur dans cette architecture peut devenir un goulot d'étranglement en cas de charge de trafic élevée, ce qui réduit les performances et la disponibilité (Stefanova-Stoyanova *et al.*, 2021).

Pair à Pair (P2P) est discutée dans 4 PSs. C'est une alternative à l'architecture Client Serveur dans laquelle chaque système a une priorité égale et est soit un serveur, soit un client (Stefanova-

Stoyanova *et al.*, 2021). Les pairs (dispositifs IoT) sont connectés directement les uns aux autres, sans intermédiaire ni composant central.

Cette architecture offre une productivité élevée et une bonne accessibilité. La flexibilité, la modifiabilité et la scalabilité sont également garanties, car les pairs peuvent être ajoutés ou supprimés du système à tout moment. Le système reste viable si l'un des points cesse de fonctionner. Il permet la conception de systèmes de gestion de l'énergie en P2P avec des fonctionnalités clés telles que la décentralisation, la transparence, la confidentialité, la scalabilité, l'efficacité énergétique et le contrôle de l'utilisateur (Mullaney, Aijaz, Sealey & Holden, 2022).

En ce qui concerne les inconvénients, l'architecture pair à pair présente un mécanisme complexe de gestion de la sécurité et de l'accessibilité du système, car il n'y a pas de composant central. Les performances et la disponibilité du système dépendent fortement des performances et de la disponibilité du réseau de transmission des données (Stefanova-Stoyanova *et al.*, 2021).

Tubes et Filtres est discutée dans 4 PSs. C'est un modèle architectural qui implique une architecture système où une série de composants de traitement, appelés filtres, sont reliés par des canaux appelés tubes pour effectuer des tâches spécifiques de manière linéaire et séquentielle. Cette architecture convient bien à l'IoT, mettant des composants modulaires qui interagissent via des pipelines de données pour un transfert de données efficace. Chaque composant remplit un rôle distinct ou agit comme un filtre pour le traitement des données dans le pipeline.

L'architecture Tubes et Filtres permet d'ajouter et de supprimer facilement des composants au besoin, ce qui la rend évolutive et adaptable à différents systèmes IoT (Barroca Filho & de Aquino Junior, 2018). Les filtres peuvent être développés de manière indépendante et réutilisés dans différents systèmes IoT (Barroca Filho & de Aquino Junior, 2018). Dans un contexte d'interopérabilité, les Tubes et Filtres peuvent composer une séquence de files d'attente de messages pouvant être utilisées pour fournir l'infrastructure nécessaire à la mise en œuvre d'un pipeline (Valle, Garcés & Nakagawa, 2019).

Cependant, ce type de traitement par lots de données dans l'IoT ne gère pas bien l'interactivité, et la latence entraîne une dégradation des performances (Tekinerdogan & Köksal, 2018).

Architecture orientée événements est discutée dans 4 PSs. C'est une architecture logicielle qui met l'accent sur la production, la détection, la consommation et la réaction aux événements. Dans le contexte de l'IoT, cette architecture est populaire par son utilisation pour créer des systèmes intelligents capables de répondre en temps réel aux événements et déclencheurs. Les dispositifs et les capteurs produisent en continu des données et des événements, qui sont traités par des moteurs de traitement d'événements ou des systèmes de traitement d'événements complexes.

Les avantages de l'architecture orientée événements incluent la possibilité de réutiliser des composants, de modifier facilement des solutions, de traiter de multiples demandes en même temps pour des performances élevées, de s'étendre indépendamment et de surveiller efficacement les données.

Comme inconvénients, nous citons la complexité des tests. Il est difficile de trouver des erreurs en raison de la nature asynchrone, le temps de réponse et de la possibilité de défaillance du système avec le bus d'événements (Stefanova-Stoyanova *et al.*, 2021).

Architecture REST est l'architecture la moins discutée avec 2 PSs. C'est un style de construction de services Web qui a été appliqué à la conception de systèmes IoT. Dans les systèmes IoT, l'architecture REST peut être utilisée pour fournir une interface standard et cohérente pour que les dispositifs et les applications interagissent les uns avec les autres. Les dispositifs IoT peuvent être modélisés comme des ressources, et les services Web REST peuvent être utilisés pour accéder et manipuler ces ressources (Zhang, Wen, Wu & Zou, 2011).

Les avantages de l'architecture REST sont la réutilisation et la scalabilité grâce à l'utilisation de services faiblement couplés. L'architecture REST améliore également l'efficacité en permettant au client de mettre en cache les informations reçues du serveur et en améliorant la capacité de traitement des demandes en utilisant des caches partagés et des mandataires inverses (Jacob & Mani, 2018). Cette architecture permet une fonctionnalité flexible en permettant

au serveur d'étendre temporairement les capacités du client grâce à l'exécution de code (Jacob & Mani, 2018).

Cependant, comme inconvénient, il peut être difficile de travailler avec l'architecture REST (Stefanova-Stoyanova *et al.*, 2021), en particulier aux premières étapes de la conception. En tant qu'architecture, elle n'est pas très adaptée à une utilisation avec des transferts de données confidentiels en raison de problèmes de sécurité.

Recommandation pour les praticiens : La liste des styles architecturaux discutés dans la littérature n'est pas exhaustive. Les architectures Microkernel, Multi-tenancy, Monolithic et Data-Centric sont quelques-unes des nombreuses architectures potentiellement non discutées dans la littérature. Bien que les architectures SOA et l'architecture en couches soient les plus importantes, nous suggérons d'explorer plus en détail l'architecture microservices car elle prend en charge des services distribués et interopérables, ce qui peut être idéal pour les grands systèmes avec une meilleure scalabilité, une isolation des erreurs et la possibilité de déployer des services de manière indépendante. Les praticiens devraient également se tenir informés des autres styles architecturaux non explorés dans la littérature existante.

Recommandation pour les chercheurs : Nous avons observé que peu de styles architecturaux sont discutés dans la littérature existante. Cependant, il existe d'autres styles architecturaux, tels que Data-Centric et Multi-tenancy, qui ne sont pas mentionnés dans nos PSs. Nous suggérons la nécessité d'étudier d'autres styles architecturaux pour les systèmes IoT.

3.2.3 QR3 : Les styles architecturaux identifiés dans la QR2 répondent-ils aux exigences de qualité énoncées dans la QR1 ?

QR3 prend en considération les styles architecturaux possibles utilisées pour les systèmes IoT telles que rapportées dans QR2 ainsi que la liste des exigences en matière de qualité qui ont été recueillies dans le cadre de QR1. Nous avons fait correspondre les réponses de QR1 et QR2 pour déterminer les possibilités qui pourraient répondre aux exigences en matière de qualité.

Tableau 3.5 Styles architecturaux et exigences de qualité

	Scalabilité	***Sécurité***	***Disponibilité***	***Interopérabilité***	***Performance***	***Confidentialité***	***Fiabilité***	***Gestion de l'énergie***	***Flexibilité***	***Évolutivité***	**Total**
Architecture basée sur le Cloud	✓	✓	×	✓	✓	×	✓	×	✓	✓	7
Architecture REST	✓	×	×	✓	✓	×	✓	✓	✓	**NA**	6
SOA	✓	×	✓	×	×	×	✓	×	✓	✓	5
Pair à Pair	✓	×	✓	×	×	×	✓	✓	✓	**NA**	5
Architecture orientée événements	×	✓	×	×	✓	×	×	✓	×	**NA**	4
Tubes et Filtres	✓	**NA**	**NA**	✓	×	**NA**	**NA**	**NA**	×	✓	3
Publish Subscribe	✓	✓	×	×	×	✓	✓	×	×	**NA**	3
Microservice	✓	×	×	✓	×	×	×	×	✓	**NA**	3
L'architecture en couches	×	✓	×	✓	×	×	**NA**	×	×	**NA**	2
Client Serveur	✓	**NA**	×	×	×	×	✓	×	×	**NA**	2
Total	7	3	2	5	3	1	6	3	5	3	

NA Information non disponible, ✓ Architecture vérifie les exigences de qualité, **×** Architecture ne vérifie pas les exigences de qualité

Certaines de ces exigences ont déjà été mentionnées dans le cadre de QR2, car elles représentent les avantages des styles architecturaux.

Le Tableau 3.5 montre la correspondance entre les styles architecturaux et les exigences en matière de qualité tels que rapportés dans les articles étudiés. Pour chaque architecture, nous avons marqué les exigences en matière de qualité par une coche "✓" si elles sont dites satisfaites par cette architecture dans les articles étudiés. En revanche, nous avons utilisé une marque "×" pour indiquer que les exigences en matière de qualité ne sont pas remplies par cette architecture spécifique selon les articles étudiés. C'est-à-dire que dans la liste des articles étudiés, nous avons

trouvé mention du fait que ces architectures ont un impact négatif ou nuisent à l'exigence de qualité correspondante. Les informations manquantes dans la littérature sont indiquées par "NA". Par exemple, dans l'ensemble des PSs, nous n'avons trouvé aucune information sur l'évolutivité du système IoT lors de l'utilisation d'une architecture de microservices.

Chaque architecture est associée à des exigences de qualité spécifiques :

1. **Scalabilité :** SOA, l'architecture microservices, l'architecture Publish Subscribe, l'architecture Tubes et Filtres, l'architecture basée sur le Cloud et l'architecture Pair à Pair offrent une scalabilité aux systèmes IoT (Mullaney *et al.*, 2022; Jacob & Mani, 2018; Pena *et al.*, 2015; Sethi & Sarangi, 2017; Tekinerdogan & Köksal, 2018). SOA atteint la scalabilité en décomposant des applications complexes en services faiblement couplés pouvant être indépendamment dimensionnés. L'architecture microservices permet la scalabilité horizontale de composants spécifiques, tandis que l'architecture Publish Subscribe désolidarise les éditeurs et les abonnés, favorisant une scalabilité efficace. L'architecture Tubes et Filtres s'adapte aux flux de données accrus en ajoutant ou en supprimant des composants de traitement des données. L'architecture Pair à Pair s'adapte en distribuant le traitement et la gestion des données entre les pairs interconnectés, ce qui permet de répondre à la croissance du réseau en ajoutant davantage de pairs pour le traitement et la distribution des données ;
2. **Sécurité :** Dans l'architecture en couches, la sécurité est une préoccupation, c'est pourquoi cette architecture propose des mécanismes pour l'échange sécurisé de données entre les objets. L'architecture Publish Subscribe assure la sécurité grâce à divers mécanismes, notamment le chiffrement des données, le contrôle d'accès et des protocoles de communication sécurisés tels que TLS/SSL. De plus, l'architecture REST ne facilite peut-être pas entièrement les exigences en matière de sécurité, mais elle offre une option pour les appareils à faible puissance en se basant sur le découplage de modules et d'objets reposant principalement sur HTTP pour la communication. L'architecture orientée événements renforce la sécurité en améliorant la détection et la réponse aux analyses d'événements de sécurité et en réduisant les attaques à la surface du système (Stefanova-Stoyanova *et al.*, 2021; Tekinerdogan & Köksal, 2018; Burhan *et al.*, 2018) ;

3. **Interopérabilité :** Les styles architecturaux REST, Tubes et Filres, Microservices, L'architecture en couches et architecture basée sur le Cloud améliorent l'interopérabilité des systèmes IoT (Stojanov & Dobrilovic, 2021; Stefanova-Stoyanova *et al.*, 2021; Tekinerdogan & Köksal, 2018; Tekinerdogan, Köksal & Çelik, 2023; Guerrero-Ulloa, Rodríguez-Domínguez & Hornos, 2023). Pour ce faire, l'architecture REST s'appuie sur des protocoles de communication normalisés et sans état tels que HTTP. L'architecture Tubes et Filtres permet l'adaptabilité et la compatibilité à mesure que les données circulent à travers une série de filtres capables de traiter différents formats. L'architecture Microservices prend en charge l'interopérabilité grâce à des services modulaires et des API normalisées. L'architecture en couches rationalise la communication en organisant les données en couches hiérarchiques. L'architecture basée sur le Cloud centralise la gestion des dispositifs, permettant à des dispositifs de capacités variables de se connecter et de communiquer via le cloud, simplifiant les interactions complexes ;
4. **Disponibilité :** SOA garantit la disponibilité en distribuant les services et en permettant aux dispositifs d'accéder et d'utiliser ces services de manière indépendante, réduisant ainsi le risque de points de défaillance uniques. L'architecture Pair à Pair offre la disponibilité grâce à la distribution des nœuds, garantissant que même si certains nœuds échouent, le réseau reste accessible. L'architecture Client Serveur assure la disponibilité en séparant les fonctions de traitement et de stockage dans des serveurs dédiés (Barroca Filho & de Aquino Junior, 2018; Guerrero-Ulloa *et al.*, 2023) ;
5. **Flexibilité :** SOA atteint la flexibilité en permettant le développement, la modification et l'intégration indépendants des services. L'architecture microservices offre la flexibilité en décomposant les applications en services pouvant être déployés de manière indépendante, permettant des mises à jour rapides et des personnalisations. L'architecture Pair à Pair favorise la flexibilité en permettant aux dispositifs de communiquer directement. L'architecture basée sur le Cloud améliore la flexibilité grâce à des ressources et des services à la demande. L'architecture REST prend en charge la flexibilité grâce à ses interfaces sans état et uniformes, simplifiant l'ajout ou la modification de services sans perturber l'ensemble du système

(Tekinerdogan & Köksal, 2018; Jacob & Mani, 2018; Pena *et al.*, 2015; Ahmed *et al.*, 2019a) ;

6. **Évolutivité :** SOA, l'architecture basée sur le Cloud et l'architecture Tubes et Filtres favorisent l'évolutivité en permettant l'intégration aisée de nouveaux composants et mises à jour. SOA atteint l'évolutivité en offrant une structure modulaire qui intègre de manière transparente de nouveaux services et composants, s'adaptant aux technologies émergentes et aux besoins évolutifs de l'entreprise. L'architecture basée sur le Cloud facilite l'évolutivité grâce à des mises à jour faciles, à la mise à l'échelle et à l'intégration de nouvelles fonctionnalités via des services cloud, garantissant une flexibilité en réponse aux demandes changeantes et aux avancées dans le domaine de l'IoT.

   L'architecture Tubes et Filtres favorise l'évolutivité dans les systèmes IoT en offrant un moyen flexible et extensible de traiter et de transformer les données, permettant l'introduction de nouveaux composants de traitement des données et l'adaptation aux formats et sources de données évolutifs dans les systèmes IoT (Meloni *et al.*, 2018; Vermeire *et al.*, 2019; Wilde, Gonen, El-Sheikh & Zimmermann, 2016) ;
7. **Performance :** L'architecture basée sur le Cloud excelle en matière de performance en offrant l'équilibrage de charge et la gestion de plusieurs événements. L'architecture REST assure un transfert efficace des données et repose sur des méthodes HTTP standard pour la communication, ce qui permet la mise en cache, la réduction des transferts de données redondants et la réduction de la latence. L'architecture orientée événements excelle dans la gestion de nombreux événements simultanément, ce qui peut améliorer les performances globales du système (Guerrero-Ulloa *et al.*, 2023; Kumar *et al.*, 2017; Tekinerdogan & Köksal, 2018; Barroca Filho & de Aquino Junior, 2018; Stefanova-Stoyanova *et al.*, 2021) ;
8. **Confidentialité :** L'architecture Publish Subscribe améliore la confidentialité dans les systèmes IoT en permettant aux utilisateurs de s'abonner uniquement aux données ou aux événements qui les intéressent, réduisant ainsi l'exposition inutile d'informations sensibles et le risque de violations de données (Guerrero-Ulloa *et al.*, 2023; Dragoni *et al.*, 2017) ;
9. **Fiabilité :** SOA améliore la fiabilité en décomposant des tâches complexes en services réutilisables, réduisant l'impact des défaillances. L'architecture Publish Subscribe améliore

la fiabilité en facilitant la distribution efficace d'événements et la tolérance aux pannes grâce à des composants redondants. L'architecture Pair à Pair contribue à la fiabilité en distribuant des tâches à travers un réseau de pairs, réduisant les risques de point de défaillance unique et garantissant la tolérance aux pannes. L'architecture basée sur le Cloud assure la fiabilité dans les systèmes IoT en offrant des services cloud évolutifs et redondants. L'architecture Client Serveur améliore la fiabilité dans les systèmes IoT en centralisant les fonctions critiques sur les serveurs, permettant la redondance et l'équilibrage de charge. Enfin, l'architecture REST améliore la fiabilité dans les systèmes IoT en utilisant une communication sans état et des méthodes HTTP standard (Ibrahim *et al.*, 2019; Stefanova-Stoyanova *et al.*, 2021; Jacob & Mani, 2018) ;

10. **Gestion de l'énergie :** L'architecture Pair à Pair optimise l'utilisation de l'énergie en partageant des tâches et des données. L'architecture REST minimise la surcharge énergétique dans les échanges de données et favorise une utilisation efficace des ressources. L'architecture orientée événements offre une gestion de l'énergie aux systèmes IoT en concevant le système de manière réactive, en ne répondant qu'aux événements nécessitant une action (Mullaney *et al.*, 2022; Javadpour *et al.*, 2020).

Recommandation pour les praticiens : L'architecture REST et basée sur le Cloud satisfont la plupart des exigences de qualité identifiées dans les PSs pour les systèmes IoT ; cependant, elle ne répond pas aux exigences de sécurité, de confidentialité et de disponibilité. Bien que l'architecture Microservices ne satisfasse que quelques exigences de qualité identifiées dans nos PSs, elle est une autre architecture recommandée en raison de sa capacité à fournir la scalabilité, l'interopérabilité et la flexibilité, qui sont essentielles pour les systèmes IoT compte tenu du grand nombre de dispositifs IoT impliqués. Les praticiens doivent évaluer les avantages et les inconvénients de chaque style architectural proposé pour identifier l'architecture qui répond le mieux aux exigences du système.

Nos conclusions pour les praticiens sont résumées comme suit :

1. Les architectures telles que REST, SOA, basée sur le Cloud et Microservices fournissent des exigences de scalabilité et de flexibilité pour répondre au développement et au déploiement indépendants de services ;
2. Les exigences de sécurité ont été négligées dans différents styles architecturaux. Les praticiens doivent prendre la bonne décision en fonction des exigences de sécurité spécifiques ;
3. La correspondance entre les styles architecturaux et les exigences de qualité peut servir de référence aux praticiens pour choisir la bonne architecture en fonction des priorités en matière d'exigences de qualité.

Recommandation pour les chercheurs : Des études supplémentaires sont nécessaires pour examiner divers styles architecturaux pour les systèmes logiciels traditionnels et évaluer leur applicabilité aux systèmes IoT. Ces études devraient évaluer minutieusement les exigences de qualité pour tous les systèmes IoT. Les recommandations de telles études pourraient inclure le développement d'une matrice complète décrivant toutes les exigences de qualité pour les systèmes IoT dans différents domaines d'application. De plus, un guide détaillé pourrait être produit pour faciliter les décisions éclairées lors de la sélection de l'architecture la plus appropriée.

3.3 Discussion

Dans cette section, nous discutons de nos résultats et de certaines implications possibles tant pour les praticiens que pour les chercheurs. Ensuite, nous explorons les styles architecturaux de l'IoT dans divers domaines et met en lumière les liens entre ces styles architecturaux et les exigences spécifiques de chaque domaine. En identifiant les structures les plus fréquemment utilisées, cette analyse offre un aperçu des choix d'architecture et de leur pertinence dans des secteurs tels que les systèmes intelligents, l'agriculture, les soins de santé, les bâtiments intelligents et les villes. De plus, nous examinons comment l'infrastructure IoT impacte et oriente le choix des styles architecturaux pour répondre aux besoins et aux limitations des solutions IoT.

3.3.1 Couverture limitée

Les PSs analysés présentent une couverture limitée des styles architecturaux et des exigences de qualité, soulignant la nécessité de recherches supplémentaires dans ces domaines. Nous discutons de différentes exigences de qualité principalement abordées dans les PSs, mais il existe de nombreuses autres exigences de qualité non discutées, telles que la connectivité, la distributivité, la dynamique et l'utilisation des ressources (Minani, Sabir, Moha & Guéhéneuc, 2024). Divers styles architecturaux identifiés dans les PSs sont discutés, mais d'autres styles architecturaux tels que Microkernel, Multi-tenancy et Data-Centric architectures Sharma, Kumar & Agarwal (2015) ne sont pas mentionnés dans l'un des PSs.

L'interopérabilité est une autre exigence de qualité importante pour les systèmes IoT compte tenu des technologies diverses utilisées pour les implémenter (Minani *et al.*, 2024). Cependant, seuls les styles architecturaux Microservice, en couches, basés sur le Cloud, REST, Publish Subscribe et Tubes et Filtres prévoient l'interopérabilité. Bien que les architectures SOA, Microservice, Publish Subscribe, basée sur le Cloud, Client Serveur semblent être populaires dans les systèmes IoT, il est étonnant que dans les PSs, l'efficacité énergétique ne soit pas considérée parmi les exigences de qualité, compte tenu du besoin d'efficacité énergétique dans les dispositifs IoT. L'évolutivité est une autre exigence de qualité que nous pensons que toute architecture de système IoT devrait fournir. Cependant, SOA, basée sur le Cloud et Tubes et Filtres satisfont cette exigence.

Bien que la sécurité semble être considérée comme satisfaite par une architecture basée sur le Cloud dans la majorité des études examinées et comme étant courante, d'autres études, par exemple celle de Kashyap, Rana, Kansal & Walia (2021), montrent que les problèmes de sécurité deviennent critiques avec une architecture basée sur le Cloud, car les appareils IoT et les utilisateurs partagent continuellement des données, des ressources de calcul et de réseau à distance. De plus, la préservation de la confidentialité des données pose également un problème dans l'IoT basé sur le Cloud. Il est stipulé aussi que l'architecture IoT basée sur le Cloud repose

sur trois couches différentes qui sont sujettes à de nombreuses attaques, telles que les attaques actives et passives.

3.3.2 Implications pratiques pour les praticiens

Il n'existe pas de style architectural universellement recommandé ou meilleur pour les systèmes IoT. Cependant, compte tenu de la nature unique de ces systèmes, comme discuté dans le travail de Minani *et al.* (2024), la scalabilité, l'interopérabilité et la flexibilité sont des exigences de qualité supérieures que nous pensons que tout système IoT devrait fournir. Par conséquent, le style architectural répondant à ces exigences de qualité pourrait être considéré comme meilleur que ceux qui n'y répondent pas. L'architecture basée sur le Cloud est également une bonne option, en supposant que la scalabilité est gérée par le fournisseur de services cloud Jin *et al.* (2022). Le style architectural Tubes et Filtres pourrait être une autre bonne option car il offre l'évolutivité, bien qu'il ne fournisse pas la flexibilité souhaitée. En fonction du domaine d'application, les praticiens évaluent à la fois les avantages et les inconvénients de chaque style architectural recommandé et décident quelle architecture convient le mieux à leurs besoins.

3.3.3 Implications pratiques pour les chercheurs

De nombreux styles architecturaux pour les systèmes logiciels traditionnels, tels que Microkernel, Multi-tenancy et l'architecture Data-Centric, ne sont discutés dans aucun des PSs. De même, certaines exigences de qualité des systèmes IoT, telles que la connectivité, la distributivité, la dynamique et l'utilisation des ressources, ne sont pas abordées dans les PSs. Nous pensons que des investigations supplémentaires sont nécessaires pour étudier de manière approfondie tous les styles architecturaux et les exigences de qualité correspondantes, évaluant comment chacun de ces styles architecturaux satisfait ces attributs de qualité dans le but de trouver le meilleur style architectural pour les systèmes IoT en général ou pour des domaines d'application spécifiques. Cela est dû au fait que certaines exigences de qualité peuvent être d'une grande importance dans un domaine d'application, mais pas dans l'autre, et vice versa.

3.3.4 Domaines de l'IoT et architectures

Le développement de l'IoT conduit à l'émergence de divers domaines tels que les systèmes intelligents, l'agriculture, les soins de santé, les bâtiments et les villes. Chaque domaine présente des exigences et des défis uniques nécessitant des architectures spécifiques pour les satisfaire. Certaines études primaires indiquent l'utilisation d'architectures spécifiques pour différents domaines IoT.

Dans les systèmes intelligents, les architectures telles que l'architecture Client-Serveur, l'architecture Pair à Pair et l'architecture en couches sont couramment utilisées.

Pour l'agriculture (Gavrilović & Mishra, 2021), l'architecture basée sur le Cloud, l'architecture Client Serveur et l'architecture Pair à Pair sont utilisées pour collecter, stocker et analyser des données provenant de capteurs et d'actionneurs. Pour les soins de santé (Muccini & Moghaddam, 2018a; Gavrilović & Mishra, 2021; Santana *et al.*, 2018), des architectures telles que l'architecture en couches, l'architecture basée sur le Cloud et l'architecture Client Serveur sont utilisées pour garantir la confidentialité et la sécurité des données médicales. Dans les bâtiments intelligents (Jia *et al.*, 2019), des architectures telles que l'architecture microservices, SOA et l'architecture en couches sont utilisées pour surveiller et contrôler les systèmes du bâtiment. Dans les villes intelligentes (Kyriazopoulou, 2015), des architectures telles que SOA, l'architecture en couches et l'architecture basée sur le Cloud sont utilisées pour connecter différents appareils IoT et fournir des services intelligents.

Concernant l'infrastructure IoT, nous la présentons dans cette section car elle influence le choix de l'architecture. L'infrastructure IoT fournit les ressources nécessaires au fonctionnement d'un système IoT, tandis que l'architecture IoT fournit le cadre conceptuel et logique pour la manière dont ces ressources travaillent ensemble. Ainsi, une architecture IoT bien conçue devrait tenir compte des capacités et des limitations de l'infrastructure IoT pour assurer une utilisation efficace des ressources tout en répondant aux exigences de qualité de la solution IoT. L'utilisation du Cloud, qui permet d'exploiter des applications avec moins de soucis pour les détails de l'infrastructure de calcul, est répandue dans les systèmes IoT. Cependant, la latence est une

préoccupation, conduisant à l'émergence du Fog Computing, qui divise les appareils et les services en zones de calcul en fonction de leurs emplacements relatifs pour traiter les données et les événements instantanément au niveau du réseau Fog. Les microservices (Pallewatta, Kostakos & Buyya, 2019) et l'architecture en couches (Magaia *et al.*, 2020) peuvent facilement adopter l'architecture Fog Computing.

3.4 Menaces pour la validité

Nous tentons de minimiser les menaces à la validité en suivant un processus étape par étape de revue de la littérature systématique proposé par Kitchenham *et al.* (2009). Cependant, la recherche empirique ne peut pas être réalisée sans limitations. Cette section discute de la manière dont nous minimisons les menaces à la validité dans la recherche décrite.

Validité Externe : Une menace potentielle à la validité externe pourrait être le manque de représentativité des PSs. Cette menace potentielle est atténuée par (1) le suivi d'une stratégie de recherche automatisée ainsi que par le snowballing avant et arrière, et (2) la définition de critères d'inclusion et d'exclusion. Nous essayons ainsi de présenter l'état de l'art et de nous assurer que le plus grand nombre d'études pertinentes possible est recherché.

Validité Interne : Pour limiter le nombre d'études manquées, nous interrogeons différentes bases de données. Le snowballing garantit également d'avoir de nombreuses études et réduit le risque de manquer d'autres études. Les résumés, par leur nature, pourraient ne pas englober des détails complets concernant les aspects spécifiques de qualité ou les éléments architecturaux discutés dans l'article. Par conséquent, cette limitation pourrait entraîner l'exclusion de PS potentiellement pertinents qui traitent de la qualité ou de l'architecture mais qui ne présentent pas de listes ou de détails explicites dans leurs résumés. Certains articles de haute qualité discutant d'aspects pertinents de l'architecture IoT ou de la qualité pourraient être omis simplement parce que leurs résumés manquent de listes spécifiques ou de discussions détaillées à cet égard. Pour atténuer cette menace potentielle, un processus de validation rigoureux est adopté dans notre SLR. Une autre menace possible est le manque d'informations provenant de la littérature grise

discutant des architectures IoT et des exigences de qualité. Nous choisissons délibérément de ne pas inclure la littérature grise en raison du nombre d'PSs incluses dans cette étude ; nous considérons donc cette menace comme acceptable.

Validité de la Conclusion : Certaines des études que nous identifions peuvent ne pas être matures et peuvent affecter la généralisabilité de la conclusion. Pour atténuer ce risque, nous sélectionnons soigneusement les études pertinentes, et validons la pertinence des données extraites pour chaque article afin de minimiser la possibilité de biais.

3.5 Conclusion

Dans ce chapitre, nous tentons de résoudre le problème relatif à l'absence d'une étude exhaustive répertoriant toutes les exigences de qualité et les styles architecturaux logiciels les plus couramment utilisés dans les systèmes IoT. De plus, aucune étude n'est entreprise pour évaluer si ces styles architecturaux favorisent efficacement la mise en œuvre, le contrôle et le respect des exigences de qualité les plus fréquemment rencontrées, ni pour garantir leur application.

Pour aborder cette lacune, nous identifions des études concernant les exigences de qualité et les styles architecturaux utilisés dans les systèmes IoT, et nous analysons et discutons des résultats rapportés. Une recherche automatique dans les bibliothèques numériques les plus pertinentes pour les études publiées jusqu'en 2023 nous permet d'identifier 20,616 études potentielles. Après le processus de sélection, nous obtenons 103 études. Nous analysons ces études pour discuter de trois questions de recherche. Ces questions explorent les exigences de qualité pour les systèmes IoT, les styles architecturaux logiciels utilisés pour concevoir les systèmes IoT, leurs avantages et inconvénients pour l'IoT, et s'ils satisfont aux exigences de qualité des systèmes IoT.

Nous identifions dix exigences de qualité que les systèmes IoT devraient satisfaire et dix styles architecturaux, et nous remarquons que tous les styles architecturaux ne fournissent pas les mêmes avantages ou ne répondent pas à toutes les exigences de qualité.

Nous évaluons dans quelle mesure ces styles architecturaux s'alignent sur les exigences de qualité, révélant que SOA, Client Serveur et REST répondent le mieux à ces critères. Cependant, les exigences en matière de confidentialité bénéficient d'un soutien limité, soulignant la nécessité d'une architecture adaptée aux besoins et contraintes spécifiques du système IoT, divers styles architecturaux excellent dans la scalabilité, l'intégration, la disponibilité, la sécurité, la flexibilité et l'adaptabilité, offrant des possibilités variées pour la connectivité et la scalabilité de l'IoT à l'ère numérique.

Dans des travaux futurs, nous pourrions mener des expériences pour comparer les styles architecturaux existants avec le système IoT en évaluant quantitativement les exigences de qualité. La maintenabilité, la compatibilité et la portabilité sont des exigences de qualité importantes des systèmes IoT, mais elles reçoivent peu d'attention dans la littérature. Plus de recherches peuvent être effectuées sur ces exigences de qualité à l'avenir.

CHAPITRE 4

COMPARAISON DE LA QUALITÉ DU CODE ET DES BONNES PRATIQUES DANS LES LOGICIELS IOT ET NON IOT

Comme indiqué précédemment, la qualité des systèmes IoT peut englober la qualité des architectures ou du code. Nous débutons par une approche à la granularité la plus large, examinant dans le chapitre précédent les architectures associées aux systèmes IoT et leurs impacts sur les exigences de qualité de ces systèmes. Par la suite, dans ce chapitre, nous explorons la qualité du code des systèmes IoT.

Comme mentionné dans le chapitre 2, de nombreuses études portent sur la qualité du code des systèmes non IoT en comparaison à ceux de l'IoT. Aussi, il est incertain que les résultats de ces études puissent être appliqués aux systèmes IoT, car il n'y a pas d'étude comparative examinant la différence au niveau du code entre les systèmes IoT et non IoT. Cela limite notre compréhension de la manière dont les systèmes IoT diffèrent des systèmes non IoT en termes d'exigences de qualité et de défis. Nous perdons également l'opportunité d'utiliser des modèles de qualité existants pour les systèmes non IoT s'ils sont similaires sur certains points, car il y a une grande maturité dans la recherche sur la qualité des systèmes non IoT. Pour ce faire, nous menons une étude empirique comparative entre le code d'un ensemble des systèmes IoT par rapport aux systèmes non IoT, suivie d'une analyse approfondie des systèmes. Compte tenu des différences de la comparaison, nous fournissons une liste des bonnes pratiques pour le développement de systèmes IoT.

Dans la suite, nous présentons une analyse quantitative de nos résultats de comparaison, suivie d'une analyse approfondie de certains systèmes IoT concernant les résultats de la comparaison qualitative. En illustrant avec des exemples spécifiques la complexité des systèmes IoT et comment elle se manifeste dans les bases de code IoT, nous abordons ensuite les implications pratiques de nos résultats pour le développement des systèmes IoT. La discussion sur les résultats des bonnes pratiques est ensuite introduite, suivie d'une section sur les menaces pour la validité de notre travail. En conclusion, nous résumons nos résultats.

4.1 Méthodologie

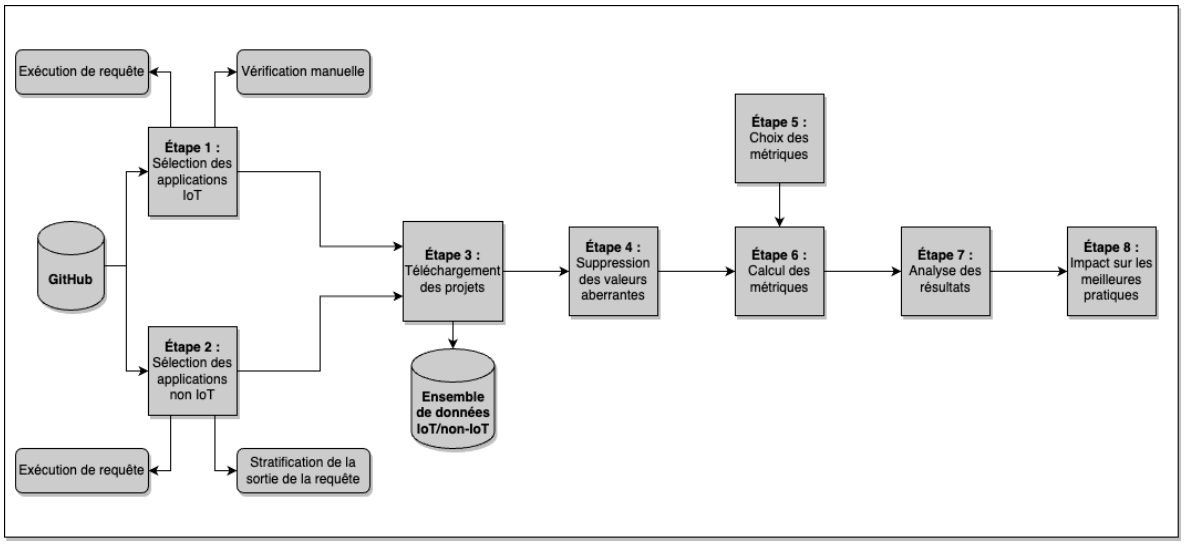

Figure 4.1 Méthodologie de recherche de la comparaison

Trouver des systèmes IoT et des systèmes non IoT comparables est un défi, étant donné la nécessité de faire correspondre des critères tels que les étoiles, les forks, la taille, le langage de programmation, le nombre des classes et des fichiers. En suivant cette approche nous n'avons pas trouvé de résultats utiles, nous avons donc adapté notre processus de sélection pour nous concentrer sur des nombres similaires d'étoiles et de forks.

La popularité d'un dépôt, telle qu'indiquée par les étoiles et les forks, peut refléter sa pertinence dans un certain domaine ou pour un cas d'utilisation particulier (Borges & Valente, 2018). Les étoiles de GitHub offrent aux utilisateurs un moyen d'exprimer leur appréciation pour les dépôts de code (Corno *et al.*, 2020). Lorsque deux dépôts partagent une popularité similaire, cela implique qu'ils sont appréciés dans leurs catégories respectives. Cette approche garantit une attention et une reconnaissance de la part de la communauté GitHub.

Notre méthodologie a été influencée par Politowski, Petrillo, Montandon, Valente & Guéhéneuc (2021) et Corno *et al.* (2020), avec huit étapes clés comme indiqué dans la Figure 4.1.

En octobre 2022, nous avons initié la première étape de notre processus en collectant des systèmes IoT populaires depuis GitHub. Nous avons commencé par filtrer les dépôts en fonction des sujets, en nous concentrant spécifiquement sur ceux relevant des sujets "IoT", "Internet des objets", "EIoT", "IIoT" et "Internet de tout" ou "Internet industriel des objets". Les sujets GitHub servent d'étiquettes pour classer les dépôts en fonction de leur objectif, de leur sujet ou de leur communauté (Corno *et al.*, 2020). Après avoir terminé le processus de filtrage, nous avons identifié avec succès les dépôts liés aux sujets IoT. Pour donner la priorité aux dépôts avec une popularité significative et des évaluations positives des utilisateurs, nous les avons triés en fonction du nombre d'étoiles qu'ils avaient reçues, rangées par ordre décroissant. Dans la sélection finale, nous avons inclus les principaux dépôts qui étaient librement accessibles. Il convient de noter qu'une partie importante des dépôts GitHub ne concerne pas le développement logiciel. Par conséquent, nous avons effectué une inspection manuelle pour exclure les dépôts sans rapport avec le logiciel (par exemple, des didacticiels, des pages de documentation) et pour appliquer certains critères de sélection définis que nous présenterons ultérieurement.

À l'étape 2, nous avons étudié et défini des requêtes basées sur les critères des systèmes IoT sélectionnés. Nous avons utilisé des techniques d'exécution de requêtes et de stratification pour sélectionner des projets non IoT comparables parmi les applications collectées.

À l'étape 3, nous avons procédé au téléchargement des projets sélectionnés dans une base de données pour une analyse ultérieure.

À l'étape 4, nous avons identifié les valeurs aberrantes par examen manuel, puis les avons éliminées de l'ensemble de données.

À l'étape 5, nous avons sélectionné une liste complète de métriques que nous avons l'intention de calculer pour les ensembles IoT et non IoT à l'étape 6.

À l'étape 7, nous avons analysé et comparé les métriques calculées pour tirer des informations significatives.

Enfin, à l'étape 8, en se basant sur nos résultats de la comparaison entre les systèmes IoT et non IoT, nous avons systématiquement rassemblé et examiné les pratiques pour les non IoT afin de les adapter au contexte de l'IoT.

Nous présentons notre méthodologie en posant une série de questions qui ont guidé notre processus de sélection pour les systèmes IoT, les systèmes non IoT, les métriques et les outils.

4.1.1 Quels artefacts utiliserons-nous comme base de comparaison ?

Nous pourrions utiliser divers artefacts logiciels, tels que la documentation, le code, les rapports de bogues, les journaux de discussion ou les journaux d'exécution (Nazar, Hu & Jiang, 2016) dans le processus de comparaison. Nous choisissons de nous concentrer sur le code source car il est la base commune de tout système logiciel décrivant son comportement et sa fonctionnalité.

4.1.2 Comment allons-nous comparer les deux ensembles de systèmes ?

Nous comparons les deux ensembles de systèmes à l'aide de métriques car elles offrent une manière quantitative et objective d'évaluer la qualité. Les métriques fournissent des valeurs numériques permettant des comparaisons directes, réduisant la subjectivité et offrant une base claire pour évaluer les forces et les faiblesses.

Nous reconnaissons que bien que les métriques soient précieuses, elles ne fournissent peut-être pas une image complète de la qualité du système. Pour mener une évaluation approfondie, des modèles de qualité robustes tenant compte de diverses dimensions et facteurs sont essentiels. Notre travail est une première étape dans la collecte d'informations vitales en mesurant et comparant des métriques de qualité. Cela contribue au développement de modèles de qualité plus complets.

4.1.3 Quelle catégorie de métriques ?

Les métriques de code sont catégorisées par des propriétés telles que la taille, la redondance, la complexité, le couplage, la couverture des tests unitaires, la cohésion, la lisibilité du code, la sécurité, l'hétérogénéité du code et la maintenabilité (Klima *et al.*, 2022). Dans notre étude, lors de la comparaison statique du code, nous avons sélectionné des métriques de différentes catégories : taille, complexité, cohésion, couplage, lisibilité du code et maintenabilité. Nous n'avons pas exploré les aspects de sécurité, qui ont fait l'objet d'une attention considérable dans les deux systèmes (Baldini *et al.*, 2016). Nous avons choisi d'exclure les catégories de couverture et d'efficacité des tests unitaires car nous nous sommes concentrés sur les aspects statiques du code. La catégorie de redondance a également été exclue car nous la considérons étroitement liée à la lisibilité du code et à la maintenabilité.

4.1.4 Quels outils utilisons-nous pour calculer les métriques ?

Il existe une diversité d'outils que nous pouvons utiliser pour calculer ces métriques. Nous avons opté pour deux outils en raison de leur utilisation fréquente (Alenezi & Almustafa, 2015) et parce qu'ils peuvent mesurer le maximum de la liste des métriques que nous avons présentée dans le Tableau 4.1. Ces outils fournissent des informations complètes sur la complexité du système, la maintenabilité et la taille, correspondant parfaitement à nos objectifs de recherche.

Scitools Understand est conçu pour aider à comprendre, évaluer et vérifier le code source (SciTools, 2021). Il prend en charge une variété de langues et offre la possibilité de mesurer diverses métriques de code.

Multimetric est une bibliothèque Python pour créer plusieurs métriques (Multimetric, 2021). Elle est conçue pour faciliter la création de métriques complexes et multidimensionnelles pouvant être utilisées dans diverses applications. La bibliothèque fournit un ensemble complet d'API et d'utilitaires ; nous utilisons l'une des API pour mesurer nos métriques. Avec Multimetric, nous pouvons rapidement créer, combiner et analyser plusieurs métriques dans une seule base de code.

En conclusion, Scitools Understand et Multimetric ont été sélectionnés en raison de leur capacité à prendre en charge plusieurs langages de programmation et à fournir une analyse complète d'une grande liste de métriques, assurant ainsi la précision de nos mesures.

4.1.5 Quelles métriques utilisons-nous ?

Il existe une longue liste de métriques de code dans les catégories présentées dans le travail de Klima *et al.* (2022), comme indiqué dans le Tableau 4.1.

Tableau 4.1 Catégories de métriques

Catégorie	Métrique
Taille	Lignes de code (LOC), Valeur de reconstruction estimée (ERV), Taille de l'interface de l'unité (UIS), Taille moyenne de l'unité (US), Nombre de composants non architecturaux (NAC), Nombre de classes et de fichiers
Complexité	Complexité cyclomatique (CC), Volume Halstead (HV), WMC-McCabe, Nombre d'enfants (NOC), Nombre d'interconnexions de choses (NTI), Profondeur de l'arbre d'héritage (NTI)
Couplage	Réponse pour classe (RFC), Couplage entre objets (CBO), Nombre d'appels entrants par module (INC)
Cohésion	Manque de cohésion des méthodes (LCOM), Cohésion conceptuelle des classes (C3), Ratio des interactions cohésives (NRCI)
Lisibilité du code	Pourcentage de commentaires / Ratio de commentaires par rapport au code (CP)
Maintenabilité	Indice de maintenabilité (MI)

Nous avons sélectionné des métriques bien connues qui pouvaient être calculées à l'aide des deux outils choisis, Understand et Multimetric. Notre objectif était de choisir des métriques qui ne s'appliquent pas exclusivement aux systèmes IoT, mais qui sont plus générales, nous permettant d'analyser et de comparer efficacement les deux types de systèmes.

Tableau 4.2 Métriques utilisées

Catégorie	Métrique	Définition	Formule
Taille	LOC	Le nombre de lignes de code source dans le système.	LOC = Nombre de lignes non vides et sans commentaires dans le code
	#Classes	Nombre de classes du système	#Classes = Nombre de déclarations de classe dans le code source
	#Fichiers	Nombre de fichiers du système	#Fichiers = Nombre total de fichiers de code source dans le projet
Complexité	CC	Calcule le nombre de chemins linéairement indépendants dans les modules du système.	$CC(m) = E - N + 2P$ Où : m, E : nombre d'arêtes, N : nombre de nœuds, et P : nombre de composants connectés
	HV	Utilisé pour mesurer la quantité de code écrit.	HV = $N \cdot \log_2(n)$ Où : $N = N1 + N2$, n : est la somme d'opérateurs uniques et d'opérandes uniques
	WMC	Mesure la somme de complexité des méthodes d'une classe.	$WMC = \sum_{i=1}^{N} CC_i$ Où : CC_i Complexité cyclomatique de la méthode locale de McCabe i, N Nombre total de méthodes locales
Couplage	RFC	Mesure le nombre de méthodes et de constructeurs différents appelés par une classe spécifique	RFC = Fan-In + Fan-Out
	CBO	Évalue le couplage entre les classes en fonction de leur utilisation en examinant les interactions entre leurs méthodes et instances	CBO = |Ccoup| Où : C_{coup} set of classes
Cohesion	LCOM	Mesure le nombre d'ensembles distincts formés par les méthodes locales d'une classe, déterminés par leur interaction avec les variables de classe	$LCOM(C) = \frac{1}{a}\left(\frac{\sum_{j=1}^{a}\mu(Aj)-m}{1-m}\right)$ Où : a le nombre de variables dans une classe C, $\mu(Aj)$ le nombre de méthodes permettant à C d'accéder à la variable Aj, m le nombre de méthodes dans C
Lisibilité du code	CP	Quantifie le niveau de documentation en mesurant la proportion de lignes de code dédiées aux commentaires	$CP = \left(\frac{N_{\text{comment}}}{LOC}\right) \times 100\%$ Où : N_{comment} est le nombre total de commentaires dans le code source
Maintenabilité	MI	Mesure la facilité de maintenance d'un logiciel. Calculé sur la base de métriques pour un système logiciel telles que HV, CC, LOC et le COM	$MI = 171 - 5.2\ln(HV) - 0.23CC - 16.2\ln(LOC) + 50.0\sin(\sqrt{246 \cdot COM})$ Où : COM le pourcentage de lignes de commentaires par module

Le tableau 4.2 présente les métriques que nous calculons à l'aide des outils choisis ainsi que leurs formules respectives. La motivation derrière le choix de ces métriques réside dans leur capacité collective à fournir des perspectives diverses sur différents aspects de la qualité du code, allant de la taille et de la complexité du système à la maintenabilité et à la lisibilité. La sélection

vise à capturer des dimensions variées contribuant collectivement à l'évaluation de la qualité logicielle.

4.1.6 Quels systèmes choisissons-nous pour la comparaison ?

Comme mentionné dans la methodologie de recherche, nous avons obtenu les ensembles de systèmes depuis GitHub. GitHub propose une grande variété d'applications qui peuvent être utilisées pour comprendre les tendances en matière de développement logiciel, de gestion de projets et les bonnes pratiques. Ici, nous présentons une sélection de systèmes IoT et non IoT.

4.1.6.1 Comment obtenir les systèmes IoT ?

Nous avons suivi la méthode présentée ci-dessus pour sélectionner les systèmes IoT. À l'étape 1 du processus présenté dans la Figure 4.1, nous procédons à une sélection manuelle basée sur un ensemble de critères. Cela nous permet de choisir des systèmes pertinents et matures pour nos objectifs de recherche afin de fournir des informations significatives.

1. Les référentiels sous les étiquettes IoT ont différentes variantes de syntaxe (Internet des objets, IoT, EIoT, IIoT, Internet industriel des objets, Internet de tout) ;
2. Les langages du référentiel sont pris en charge par les outils d'analyse utilisés (Java, JavaScript, C, C++, C#, Python) ;
3. Le nombre d'étoiles est supérieur à 200 ;
4. Le nombre de forks est supérieur à 20 ;
5. Un référentiel actif avec la dernière mise à jour datant d'au moins 6 mois (Date de la dernière mise à jour supérieure à 04-2022) ;
6. Un référentiel mature créé entre 2012 et 2022.

4.1.6.2 Comment obtenir les systèmes non IoT ?

Nous utilisons le même ensemble de critères qui ont été utilisés pour sélectionner les systèmes IoT pour choisir l'ensemble des systèmes non IoT. Nous construisons une requête pour choisir un ensemble de systèmes non IoT avec les mêmes critères que les systèmes IoT sélectionnés. Cette requête est élaborée pour identifier les dépôts sur GitHub répondant à certains critères temporels, de popularité et technologiques basés sur notre sélection de systèmes IoT. L'objectif est de s'assurer que les dépôts sélectionnés étaient récents, populaires, activement entretenus et développés dans des langages pertinents pour notre étude, nous permettant ainsi d'analyser de nouveaux projets bien soutenus dans le domaine non IoT.

Nous construisons une requête pour choisir un ensemble de systèmes non IoT avec les mêmes critères que les systèmes IoT sélectionnés.

Notre requête est la suivante :
created : ≥ 2012-10-01 created : ≤ 2022-10-01 stars : ≥ 200 stars : ≤ 6500 forks : ≥ 20 forks : ≤ 20216 pushed : ≥ 2022-04-01 language : C language : C# language : C++ language : Java language : JavaScript language : Python

4.1.6.3 Exhaustivité de nos requêtes de recherche

Pour garantir l'exhaustivité de nos résultats, nous avons utilisé une démarche de sélection des systèmes IoT et non IoT systématique utilisant un ensemble de critères précis et variés pour identifier les dépôts pertinents sur GitHub. Les critères énumérés couvrent divers aspects des projets logiciels, notamment la syntaxe des étiquettes (tags), les langages de programmation, le niveau d'activité et de popularité des dépôts, ainsi que leur maturité pour assurer une couverture large et équilibrée des dépôts GitHub.
Cependant, malgré l'approche systématique adoptée, il est possible que certains dépôts importants aient été manqués en raison de diverses raisons. Par exemple, les étiquettes (tags) utilisées pour identifier les dépôts IoT peuvent varier en syntaxe et peuvent être sujettes à des erreurs d'orthographe ou à des variantes inattendues, ce qui pourrait entraîner la non-inclusion de

certains dépôts pertinents. De plus, les critères de popularité et d'activité, tels que le nombre d'étoiles et de forks, peuvent ne pas toujours refléter précisément l'importance ou la qualité d'un dépôt, ce qui pourrait conduire à exclure certains dépôts significatifs.
Ainsi, malgré les efforts déployés pour garantir l'exhaustivité des résultats, il est important de reconnaître les limites potentielles de la méthodologie utilisée et d'être conscient que certains dépôts importants peuvent avoir été omis. Cependant, en mentionnant les critères utilisés et en discutant des possibilités de limitations, cette approche renforce la transparence et la rigueur de notre étude, ce qui permet une évaluation critique des résultats obtenus.

4.1.6.4 Stratification de la sortie de la requête

L'exécution de la requête retourne un grands nombre de dépôts. À partir de cette sortie, nous sélectionnons un ensemble de données représentatif des systèmes IoT en utilisant la stratification.

La stratification est le processus de division d'un ensemble de données en sous-groupes homogènes basés sur certains critères. Cette approche permet une analyse plus approfondie au sein de chaque sous-groupe et contribue à garantir que les ensembles de données utilisés pour la comparaison (systèmes IoT et non IoT) soient aussi comparables que possible. Pour stratifier les données résultant des systèmes non IoT en fonction des critères représentés par les systèmes IoT, nous avons suivi ces étapes :

1. Nous extrayons des détails pertinents des référentiels IoT pour servir de critères de stratification. Nous avons choisi la composition du langage de programmation, le nombre d'étoiles et de forks. En considérant ces facteurs, nous visons à créer un sous-ensemble de systèmes non IoT qui ressemble étroitement aux caractéristiques de l'ensemble d'origine ;
2. Nous créons une correspondance entre les critères et les noms de référentiels avec chaque critère de stratification. Par exemple, un dictionnaire fait correspondre les langages de programmation à des listes de référentiels utilisant ce langage. Nous faisons la même chose avec les critères de stratification ;

3. Nous divisons les référentiels non IoT en strates en fonction des informations pertinentes. Par exemple, si nous utilisons le langage de programmation comme critère, nous créons une liste de sous-groupes, chacun contenant des référentiels utilisant le même langage de programmation ;
4. Pour chaque sous-groupe, nous sélectionnons les référentiels qui correspondent le plus aux critères représentés par les référentiels GitHub IoT. Nous utilisons le principe de Pareto (Dunford, Su & Tamang, 2014) pour sélectionner les x meilleurs référentiels, où x est le nombre de systèmes IoT sélectionnés, dans chaque sous-groupe ;
5. Nous combinons les référentiels sélectionnés dans chaque sous-groupe pour former un sous-ensemble représentatif des données ayant le même nombre et les mêmes caractéristiques que les systèmes IoT.

4.1.6.5 Comment analyser et vérifier les deux ensembles ?

À cette étape, nous analysons manuellement les deux ensembles sélectionnés pour nous assurer qu'ils ont un nombre comparable d'étoiles, de forks, et utilisent le même langage de programmation, éliminant ainsi les facteurs externes qui pourraient affecter les résultats. Pour examiner la distribution des données, nous effectuons des tests statistiques, y compris le test de Shapiro-Wilk introduit par Hanusz, Tarasinska & Zielinski (2016). Nous comparons le nombre d'étoiles et de forks dans les deux ensembles de données et utilisons le test U de Mann-Whitney non paramétrique (MacFarland, Yates, MacFarland & Yates, 2016) pour déterminer s'ils sont significativement différents. Une valeur de statistique U inférieure à $\approx$ 0.05 indique des différences significatives, tandis qu'une statistique U plus élevée suggère une comparabilité entre les ensembles de données.

4.1.6.6 Comment identifier les valeurs aberrantes ?

Les valeurs aberrantes sont des points de données qui diffèrent significativement de la majorité des données (Hawkins, 1980). Elles peuvent avoir un impact sur les résultats des analyses

statistiques. Nous éliminons les valeurs aberrantes de notre ensemble de données pour améliorer la précision de notre analyse des résultats.

Nous utilisons une approche méticuleuse pour détecter les valeurs aberrantes au sein de notre ensemble de données. Notre méthodologie a privilégié l'inspection visuelle, une technique reconnue pour l'identification des valeurs aberrantes. À travers la représentation visuelle, en particulier en traçant les données, nous avons cherché à repérer les observations qui s'écartaient notablement de la plage attendue. En évaluant systématiquement les valeurs aberrantes et leur impact potentiel sur notre analyse, nous cherchons à maintenir l'intégrité et la précision de notre ensemble de données. La suppression de ces valeurs aberrantes influentes a permis d'obtenir des analyses statistiques plus fiables et précises.

4.1.7 Comment nous avons obtenu les bonnes pratiques pour les systèmes non IoT ?

Les bonnes pratiques désignent un ensemble de lignes directrices, d'approches, de méthodes, d'outils ou de techniques recommandées et considérées comme optimales pour réduire les problèmes ou améliorer la qualité globale du logiciel.

Nous abordons l'identification de ces bonnes pratiques de manière systématique, en initiant le processus par la formulation de requêtes de recherche adaptées à chaque catégorie de métriques et en incorporant des mots-clés pertinents. La structure de la requête est conçue comme suit : **(X or Y) AND (software) AND (best practices)**, où X représente le nom de la catégorie et Y se rapporte à la pratique spécifique associée à la réduction ou à l'amélioration de la catégorie. Par exemple, dans le contexte de la catégorie de taille du code, la requête prenait la forme de **((code size) OR (code reduction)) AND (software) AND (best practices)**. Ensuite, nous exécutons ces requêtes sur Google Scholar, obtenant des nombres variables d'articles pour chaque catégorie. Ensuite, nous utilisons un processus de sélection systématique impliquant la filtration des dix articles les plus cités qui fournissaient des informations sur les bonnes pratiques pour chaque catégorie. Nous étudions et extrayons les bonnes pratiques pertinentes de ces articles. Nous évaluons et classons ces pratiques extraites en trois groupes : *directement*

applicables, *partiellement applicables avec des adaptations nécessaires*, ou *non applicables à l'IoT*. Cette catégorisation est guidée par des exigences spécifiques à l'IoT, et les pratiques ont ensuite été priorisées en fonction de leur pertinence et de leur impact potentiel sur les catégories de métriques.

4.1.8 Comment avons-nous assuré la reproductibilité de notre sélection et la généralisabilité de nos résultats ?

Nous comparons des systèmes IoT et non IoT en choisissant le sous-ensemble le plus vaste possible selon nos critères, pensant qu'il représentait bien les deux types. Notre processus de sélection a pris en compte divers langages de programmation et types de systèmes pour assurer la diversité. Cette approche s'est étendue à la sélection systématique des bonnes pratiques, garantissant la reproductibilité et la validité. La pertinence de notre ensemble de données des systèmes IoT et non IoT découle de la sélection méthodique de systèmes divers et de l'utilisation de critères stricts pour assurer la crédibilité et la généralisabilité. En choisissant attentivement des systèmes sur GitHub et en utilisant des techniques de stratification, nous avons assuré la similarité et la représentativité entre les ensembles IoT et non IoT. Les analyses statistiques ont renforcé la comparabilité, consolidant la crédibilité de nos résultats. Bien que nous n'ayons pas pu couvrir tous les systèmes, notre processus méticuleux permet des généralisations raisonnables à des contextes plus larges.

4.2 Analyse quantitative

Afin d'assurer la reproductibilité de ce travail, nous sauvegardons le code et le processus de sélection dans un paquet de réplication (www.zenodo.org/record/8078623).

4.2.1 Résultats des requêtes de recherche des systèmes IoT et non IoT

Nous exécutons nos requêtes et appliquons le processus de sélection présenté dans la section 4.1.6.1. L'exécution de notre requête sur les systèmes IoT a retourné 323 dépôts. Nous supprimons

10 dépôts en double. Ensuite, sur la base de la vérification manuelle de nos critères de sélection, nous choisissons 94 dépôts.

Nous sélectionnons 94 dépôts comparables en utilisant la technique de stratification, présentée dans la section 4.1.6.2, parmi les 1972 dépôts trouvés lors de l'exécution de la requête de recherche sur les systèmes non IoT.

4.2.2 Analyse statistique des deux ensembles de données

Nous étudions l'ensemble des données statistiques des systèmes collectées afin de comprendre leur distribution et de vérifier que les systèmes collectés sont comparables en termes de nombre d'étoiles et de forks.

4.2.2.1 Nature de la distribution

Les résultats du test de Shapiro-Wilk indiquent que la valeur p calculée est inférieure au niveau de signification de 0,05, nous avons suffisamment de preuves pour rejeter l'hypothèse nulle. Cela signifie que le nombre d'étoiles et de forks ne suit pas une distribution normale.

4.2.2.2 Le test U de Mann-Whitney entre les deux ensembles de données concernant les valeurs des étoiles et des forks

Les résultats du test U de Mann-Whitney n'ont révélé aucune différence significative entre les deux distributions d'étoiles et de forks.

4.2.2.3 Distribution des étoiles et des forks dans les deux ensembles de données

La Figure 4.2 montre le graphique de dispersion de la relation entre les étoiles en fonction des forks. La dispersion des points de données dans le graphique de dispersion donne un aperçu de la variabilité du nombre d'étoiles et de forks. La plupart des points de données sont étroitement regroupés, ce qui signifie que le nombre d'étoiles et de forks est similaire dans les deux ensembles de données.

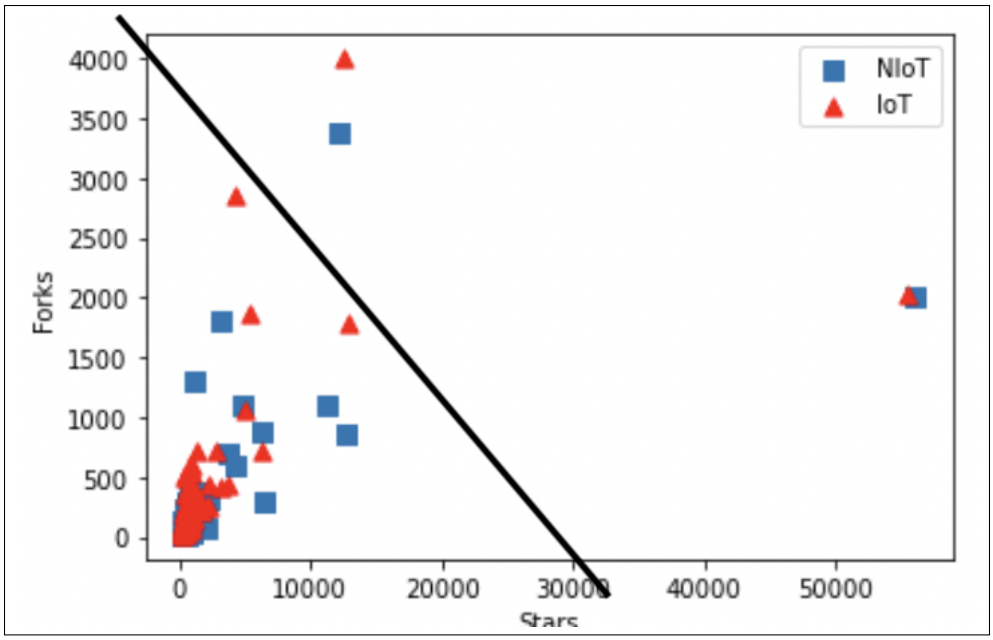

Figure 4.2 Distribution des étoiles et des forks dans les deux ensembles de données

4.2.2.4 Distribution des langages

Nous sélectionnons 14 systèmes Java, 11 C++, 16 JavaScript, 24 C, 25 Python, 4 systèmes C#.

4.2.3 Valeurs aberrantes

L'inspection visuelle a identifié des valeurs aberrantes, visibles dans la Figure 4.2, notamment thingsboard/thingsboard et home-assistant/core pour les systèmes IoT, et Apache/Druid et ansible/ansible pour les systèmes non IoT. Nous examinons leur impact sur les résultats en sélectionnant les bonnes valeurs de métriques et en les associant aux systèmes respectifs (Tableau 4.3). Apache/Druid a eu un effet significatif, nous conduisons à le retirer ainsi que son homologue IoT, thingsboard/thingsboard, de l'analyse ultérieure.

Tableau 4.3 Valeur maximale pour chaque métrique avant la suppression des valeurs aberrantes

	IoT		**non IoT**	
	Nom du Projet	**Valeur**	**Nom du Projet**	**Valeur**
RFC	Samsung/TizenRT	475185.00	Apache/Druid	191718.00
CBO	eclipse-ditto/ditto	83212.00	Apache/Druid	22423.00
CC	espressif/esp-mqtt	167.00	quarnster/SublimeGDB	339.00
HV	Samsung/TizenRT	7539487.09	quarnster/SublimeGDB	137964.39
MI	eclipse-ditto/ditto	368880.47	Apache/Druid	475261.25
LOC	Samsung/TizenRT	2009696.00	Apache/Druid	1003619.00
WMC	project-chip/connectedhomeip	83347	Apache/Druid	16057.00
LCOM	rwaldron/johnny-five	0.95	rthenica/ffmpeg-kit	0.93
CP	ARMmbed/mbed-os	70251.58	Apache/Druid	4705.60

Tableau 4.4 Valeur maximale pour chaque métrique après la suppression des valeurs aberrantes

	IoT		**non IoT**	
	Nom du Projet	**Valeur**	**Nom du Projet**	**Valeur**
RFC	Samsung/TizenRT	475185.00	DarthFubuMVC/fubumvc	64016.00
CBO	eclipse-ditto/ditto	83212.00	DarthFubuMVC/fubumvc	22423.00
CC	flomesh-io/pipy	75256385.40	mgba-emu/mgba	26505.00
HV	greghesp/assistant-relay	86796.00	dachev/node-cld	240076813.40
MI	eclipse-ditto/ditto	368880.47	wmira/react-icons-kit	811513.00
LOC	Samsung/TizenRT	2009696.00	dachev/node-cld	551449.00
WMC	project-chip/connectedhomeip	83347.00	rthenica/ffmpeg-kit	78624.00
LCOM	rwaldron/johnny-five	0.95	rthenica/ffmpeg-kit	0.93
CP	ARMmbed/mbed-os	70251.58	UnknownShadow200/ClassiCube	1413.95

4.2.4 Valeur maximale pour chaque métrique

Le Tableau 4.4 présente les valeurs de métriques logicielles les plus élevées pour les projets IoT et non IoT après la suppression des valeurs aberrantes. Les projets IoT présentent des valeurs de métriques plus élevées dans des métriques comme CC, RFC, LOC, WMC et CBO par rapport aux projets non IoT. Nous analysons ces projets et constatons que les projets IoT impliquent des interactions matérielles et logicielles plus complexes, dictées par des besoins de traitement en temps réel. Les projets non IoT présentent généralement des valeurs de métriques HV et MI plus élevées (Tableau 4.4). Cette différence s'explique par le fait que les projets non IoT sont généralement moins complexes, influencés par des pratiques de conception et de codage distinctes dans le développement logiciel non IoT.

4.2.5 Calcul statistique des métriques

Les métriques révèlent que les systèmes IoT présentent un code plus vaste et plus complexe que les systèmes non IoT en raison de leurs contraintes matérielles, nécessitant des bases de code plus importantes. Cela met en lumière les caractéristiques uniques de l'IoT et la nécessité de les prendre en compte dans la recherche et l'analyse. Cependant, la normalisation des métriques pour tenir compte de ces disparités peut entraîner une perte d'informations cruciales.

Le Tableau 4.5 indique que les systèmes IoT présentent une plus grande interdépendance que les systèmes non IoT. La moyenne de CBO de l'IoT est de 4252.41, contre 715.72 pour les projets non IoT, ce qui illustre une plus grande interconnexion dans les systèmes IoT. Cette interdépendance rend les systèmes IoT plus difficiles à entretenir et à modifier, comme en témoignent les valeurs de MI, avec une médiane de 24854.31 pour l'IoT et 36403.86 pour les systèmes non IoT.

De plus, les systèmes IoT comportent plus de classes et de fichiers par rapport aux systèmes non IoT. Par exemple, les systèmes IoT ont une médiane de 116,5 classes, tandis que les systèmes non IoT en ont 33. Cela s'explique par la nature distribuée de l'IoT, intégrant des technologies avancées et s'adaptant à diverses normes de dispositifs.

4.3 Analyse qualitative

Nous menons une étude approfondie de systèmes par paires, l'un appartenant à l'IoT et l'autre au non IoT, écrits dans le même langage. Cette exploration vise à confirmer les observations que nous avons tirées de la Tableau 4.5. Nous présentons dans la Tableau 4.6 les systèmes que nous avons analysés. Comme nous avons obtenu des résultats similaires pour chaque langage de programmation, nous avons décidé de présenter uniquement les deux systèmes développés en Java.

		IoT	non IoT
CBO	**Médiane**	**188.00**	40.00
	Moyenne	4252.41	715.72
RFC	**Médiane**	**505.00**	98.00
	Moyenne	22334.82	2956.80
LOC	**Médiane**	**4574.00**	3713.00
	Moyenne	62960.13	31374.58
WMC	**Médiane**	**217.00**	22.00
	Moyenne	4550.46	1667.51
CC	**Médiane**	**462032.02**	465.00
	Moyenne	26953.31	2358.08
HV	**Médiane**	2016.00	**283436.35**
	Moyenne	7283.76	5334.77
MI	**Médiane**	3645.21	**4666.48**
	Moyenne	24854.31	36403.86
LCOM	**Médiane**	**0.72**	0.75
	Moyenne	0.70	0.74
CP	**Médiane**	**44.87**	17.91
	Moyenne	2200.13	168.49
#Classes	**Médiane**	**116.50**	33.00
	Moyenne	975.89	373.76
#Fichiers	**Médiane**	**115.00**	47.50
	Moyenne	984.00	356.24

Tableau 4.5 Comparaison des métriques entre les deux ensembles de données

Tableau 4.6 Systèmes analysés en profondeur pour chaque langage

Langage	IoT	non IoT
Java	eclipse-ditto/ditto	kymjs/CJFrameForAndroid
JavaScript	rwaldron/johnny-five	uuidjs/uuid
C	timmbogner/Farm-Data-Relay-System	unbit/spockfs
C++	project-chip/connectedhomeip	zeek/zeek
C#	renode/renode	madskristensen/MiniBlog
Python	DT42/BerryNet	JohnHammond/msdt-follina

4.3.1 Définition des répertoires analysés

Les deux systèmes Java que nous présentons sont eclipse-ditto/ditto pour l'IoT et kymjs/CJFrameForAndroid pour le non IoT.

Eclipse-ditto/ditto (eclipse ditto, 2021) est une structure de gestion des jumeaux numériques. Un jumeau numérique est une représentation virtuelle d'un objet ou d'un système physique, et Ditto fournit un moyen de gérer les données associées à ces représentations virtuelles. Le référentiel Ditto est conçu pour prendre en charge la connectivité des dispositifs, la modélisation des données, le contrôle d'accès, le traitement des événements et l'analyse.

Kymjs/CJFrameForAndroid kymjs (2021) est un référentiel open-source destiné aux développeurs Android, fournissant un cadre pour la construction d'applications Android. Le cadre est conçu pour simplifier et accélérer le développement d'applications Android. Le cadre fournit une architecture pour la construction d'applications Android.

4.3.2 Comparaison des classes et des fichiers

Lors de la comparaison des classes les plus complexes des systèmes IoT et non IoT, nous avons constaté que les systèmes IoT ont plus de classes très complexes. La classe la plus complexe du système IoT a une complexité de 23, alors que dans le système non IoT, elle est de seulement 7.

De plus, le plus grand fichier du système IoT dépasse 2500 lignes de code, tandis que le plus grand fichier du système non IoT dépasse 600 lignes. De plus, la plus grande fonction du système IoT contient 290 lignes de code, plus grande que la plus grande fonction du système non IoT, qui contient 65 lignes.

Nous avons essayé de comprendre l'origine de ces résultats en analysant le code, et nous avons trouvé que ces différences découlent de la nature distinctive des systèmes IoT, caractérisée par la compatibilité matérielle complexe, l'intégration de capteurs, le traitement en temps réel des données, divers protocoles de communication, une gestion extensive des données et une logique métier personnalisée.
En examinant de plus près ces différences, les métriques révèlent que les systèmes IoT présentent un code plus vaste et plus complexe que les systèmes non IoT en raison de leurs contraintes matérielles, nécessitant des bases de code plus importantes. Cela met en lumière les caractéristiques uniques de l'IoT et la nécessité de les prendre en compte dans la recherche et l'analyse.

En effet, le Tableau 4.5 indique que les systèmes IoT présentent une plus grande interdépendance que les systèmes non IoT. La moyenne du CBO pour l'IoT est de 4252.41, contre 715.72 pour les projets non IoT, ce qui illustre une plus grande interconnexion dans les systèmes IoT. Ces systèmes sont caractérisés par une compatibilité matérielle complexe, une intégration de capteurs, un traitement en temps réel des données et une gestion extensive des données, cela explique la valeur CBO élevé, car de nombreuses classes ou objets dépendent les uns des autres pour fonctionner efficacement.
Cette interdépendance rend les systèmes IoT plus difficiles à entretenir et à modifier, comme en indiquant les valeurs de MI, avec une médiane de 24854.31 pour l'IoT et 36403.86 pour les systèmes non IoT.
De plus, les systèmes IoT comportent plus de classes et de fichiers par rapport aux systèmes non IoT. Par exemple, les systèmes IoT ont une médiane de 116,5 classes, tandis que les systèmes non IoT en ont 33. Cette observation s'explique par la nature distribuée de l'IoT, intégrant des technologies avancées et s'adaptant à diverses normes de dispositifs.
Une autre différence significative réside dans la complexité du traitement des données collectées à partir de différents capteurs IoT. Les modules implémentent des fonctions pour analyser, filtrer, regrouper et transformer les relevés des capteurs, nécessitant ainsi de nombreuses classes aux fonctionnalités différentes, ce qui augmente la valeur globale de RFC. Aussi, cette complexité se traduit également par une valeur de CC élevée en raison des algorithmes complexes et des API diverses utilisées dans les systèmes IoT pour l'interaction avec les appareils et les protocoles de communication tels qu'AMQP, MQTT et Apache Kafka.

4.3.3 Comparaison d'autres métriques

Le Tableau 4.7 montre les différences entre les métriques mesurées. En examinant le code, nous constatons que le RFC élevé dans le référentiel eclipse-ditto/ditto est dû à de nombreuses classes fortement couplées. Le projet gère des données et des protocoles IoT complexes, avec une classe ayant 55 importations complexes pour les connexions au service de passerelle. Le code comprend des modules qui gèrent le traitement des données collectées à partir de différents

	IoT eclipse-ditto/ditto	non IoT kymjs/CJFrameForAndroid
#Étoiles	414.00	412.00
#Forks	147.00	157.00
#Classes	**7573.00**	32.00
#Fichiers	**4917.00**	25.00
RFC	**130215.00**	328.00
CBO	**83212.00**	254.00
CC	**17056558.06**	94.00
HV	13423.00	**72196.47**
MI	368880.47	2043.00
LOC	**363467.00**	2040.00
WMC	**41499.00**	238.00
LCOM	0.62	0.67
CP	5702.18	17.23

Tableau 4.7 Comparaison des métriques mesurées

capteurs IoT. Il existe des fonctions pour analyser, filtrer, regrouper et transformer les relevés des capteurs. Le référentiel fournit également des implémentations de protocoles de communication couramment utilisés dans les systèmes IoT.

Cela nécessite de nombreuses classes aux fonctionnalités différentes, ce qui augmente la valeur globale de RFC.

Nous examinons un système avec une grande hiérarchie de classes, entraînant un couplage élevé et une valeur CBO élevée. Le code présentait également une logique complexe et des règles métier complexes, nécessitant une interaction étendue entre les objets, augmentant davantage la valeur CBO et indiquant une faible cohésion (comme le montre LCOM).

Eclipse-ditto/ditto affiche une métrique CC élevée en raison de ses algorithmes complexes, de la gestion du flux de travail et des API diverses pour l'interaction avec les appareils, les protocoles de l'appareil et les schémas de communication tels qu'AMQP, MQTT et Apache Kafka. Il présente une structure hautement modulaire, contribuant à une valeur WMC élevée.

En revanche, Kymjs/CJFrameForAndroid met l'accent sur un faible CBO en utilisant des techniques telles que l'architecture Modèle-Vue-Présentateur pour découpler le code. Ce framework se concentre également sur la réduction du volume de code tout en maintenant la fonctionnalité.

Le projet IoT possède une quantité de code plus complexe et plus importante, avec des valeurs plus élevées dans la plupart des métriques. Les projets IoT impliquent souvent l'intégration de plusieurs composants matériels et logiciels et la gestion de la communication de données entre eux, ce qui peut entraîner un plus grand nombre de classes, de fichiers et de lignes de code, comme le montre le Tableau 4.5.

En outre, les projets IoT nécessitent des algorithmes et des structures de données plus complexes pour gérer les données provenant de différents capteurs et appareils.

Les projets non IoT ont une HV plus élevée car ils utilisent plus d'opérateurs et d'opérandes distincts par rapport au projet IoT, comme indiqué dans le Tableau 4.5.

4.4 Analyse approfondie du code

Dans les sections précédentes, nous avons affirmé que les projets IoT impliquent des interactions matérielles et logicielles plus complexes, contribuant à une complexité accrue du code. L'objectif de cette section est de fournir des exemples spécifiques de chacun des systèmes IoT analysés en profondeur pour illustrer cette complexité et comment elle se manifeste dans les bases de code des systèmes IoT.

4.4.0.1 Système IoT en Java

Comme discuté précédemment, le système Java **Eclipse-ditto/ditto** est complexe.

Pour analyser davantage l'étendue de la complexité, nous avons sélectionné une classe nommée **ImplicitThingCreationMessageMapper**, qui appartient au paquet org.eclipse.ditto.connectivity.

service.mapping (www.github.com/eclipse-ditto/ditto/tree/master/connectivity/service/src/main/java/org/eclipse/ditto/connectivity/service/mapping).

Cette classe est responsable de l'intégration de nouveaux appareils ou objets IoT dans la plateforme IoT Eclipse Ditto, gérant les configurations nécessaires, les politiques et les transformations de messages pour une intégration et une gestion transparentes des appareils, ce qui rend la base de code plus complexe.

La figure 4.3 contient des configurations et une logique pour les transformations de messages. Le code utilise l'expression lambda en Java de manière imbriquée pour définir des configurations pour le mappeur de messages. Ensuite, elle est utilisée avec **l'opérateur de référence de méthode** pour filtrer et mapper la configuration de l'en-tête, augmentant ainsi la complexité du code.

```
thingTemplate = configuration.findProperty
(THING_TEMPLATE).orElseThrow(() ->
↪  MessageMapperConfigurationInvalidException
    .newBuilder(THING_TEMPLATE).build());
commandHeaders = configuration.findProperty(
↪  COMMAND_HEADERS, JsonValue::isObject,
    JsonValue::asObject).filter(configuredHeaders -> !
    ↪  configuredHeaders.isEmpty())
    .map(configuredHeaders -> { /* ... */ })
    .orElseGet(() -> DittoHeaders.newBuilder() /* ... */
    ↪  );
```

Figure 4.3 Code pour la transformation Msg dans un système IoT basé sur Java

Le code traite également de la création et de la manipulation d'entités IoT telles que Thing, Policy, ThingId, etc., comme le montrent des méthodes telles que **getCreateThingSignal**, **createInlinePolicyJson** et **validateThingEntityId**. Nous présentons quelques détails de ces méthodes dans la figure 4.4. Le code contient plusieurs importations, plusieurs interfaces et méthodes spécifiques à la plateforme IoT Eclipse Ditto, ce qui donne lieu à une base de code

complexe. Comme indiqué dans l'exemple précédent, l'environnement imbriqué intègre divers expressions, contribuant à la complexité globale du code.

```
privateSignal<CreateThing>getCreateThingSignal(finalExternalMessage
 ↪  message, final String template){}
Logic for creating a Thing based on the message and template
 ... there are other similar methods handling IoT entities
```

Figure 4.4 Code pour la création et la manipulation d'entités IoT dans un système IoT basé sur Java

4.4.0.2 Système IoT en JavaScript

rwaldron/johnny-five est un framework de programmation IoT et robotique basé sur des protocoles. Nous avons analysé le fichier eg/nodeconf-radar.js situé dans (www.github.com/rwaldron/johnny-five/blob/main/eg/nodeconf-radar.js) qui assure une visualisation de type radar avec un mouvement de balayage simulé et une détection de distance à l'aide de composants matériels, ainsi qu'une transmission de données en temps réel vers une interface web.

Le code ci-dessous dans la figure 4.5 est complexe et nécessite des experts pour le comprendre, car il contient des interactions et l'initialisation de composants matériels, ce qui nécessite une compréhension de la configuration des broches et des spécifications de plage pour le moteur servo et le capteur Ping à partir de la bibliothèque 'johnny-five'. Le code contient des nombres fictifs, par exemple, "pin", qui a une valeur de 12. De plus, la plage est définie entre 0 et 170, mais il n'est pas clair quelles fonctions ces nombres exécutent.

```
var scanner = new five.Servo({ pin: 12, range: [0, 170] });
var ping = new five.Ping(7);
```

Figure 4.5 Code JavaScript pour les interactions entre les composants matériels dans le système IoT

Le fichier contient aussi la gestion des données en temps réel avec les connexions WebSocket (Socket.io), comme illustré dans la figure 4.6. Cet extrait de code est utilisé pour configurer Socket.io afin de permettre une communication en temps réel. L'utilisation de Socket.io indique la mise en œuvre de la transmission de données en temps réel, permettant la communication entre le matériel et l'interface web via des connexions WebSocket, ce qui rend la base de code plus complexe par rapport aux systèmes non IoT. Une base de code complexe est moins efficace en termes d'utilisation des ressources et est difficile à reproduire.

```
var socket = require("socket.io");
var io = socket.listen(app);
```

Figure 4.6 Code JavaScript pour intégrer la gestion des données en temps réel avec les connexions WebSocket dans les systèmes IoT

Une autre preuve de complexité du code est qu'il contient des opérations concurrentes connu comme un facteur augmentant la complexité. Comme le montre le code dans la figure 4.7, qui assure la gestion simultanée du balayage du servo et des données du capteur Ping au sein de rappels basés sur des événements, présentant des opérations simultanées dans le rappel 'board.on("ready", function() ...)'.

```
this.loop(100, function(){}
Logic for scanning servo motion concurrently);
io.sockets.on("connection", function(socket){}
Event-driven handling of Ping sensor data while serving socket
 ↪  connections
ping.on("data",function(){}
Handling Ping sensor data concurrently);});
```

Figure 4.7 Code JavaScript pour démontrer les opérations simultanées dans les systèmes IoT

4.4.0.3 Système IoT en C

Le système **timmbogner/Farm-Data-Relay-System** utilise ESP-NOW, LoRa, et d'autres protocoles pour transporter des données de capteurs dans des zones éloignées sans dépendre du WiFi. Il est utilisée pour des scénarios nécessitant une communication à longue portée et faible consommation d'énergie entre les dispositifs IoT. Le code présente une complexité élevée en raison de plusieurs facteurs. Les exemples de code suivants sont extraits du fichier fdrs_gateway_lora.h situé dans (www.github.com/timmbogner/Farm-Data-Relay-System/blob/main/src/fdrs_gateway.h).

Le code dans le fichier examiné est complexe car il gère la communication LoRa qui implique plusieurs aspects tels que la fréquence, les niveaux de puissance, le délai d'ACK, et les tentatives de transmission, qui tous contribuent à la configuration de la radio pour la communication.

La figure 4.8 présente un exemple de code définissant des constantes pour les paramètres de configuration LoRa tels que la fréquence, et la puissance de transmission. Configurer ces paramètres est crucial pour une communication efficace, mais cela ajoute de la complexité en raison de leur variété et de leurs valeurs spécifiques.

```
#define GLOBAL_LORA_FREQUENCY 915
Setting the LoRa frequency
#define GLOBAL_LORA_SF 12
Configuring spreading factor
#define GLOBAL_LORA_TXPWR 17
Setting LoRa transmission power
...
...
(other configuration constants)
```

Figure 4.8 Code C pour définir les configurations pour LoRa

Dans l'exemple du code dans la figure 4.9, la fonction 'transmitLoRa' gère la construction et la transmission des paquets LoRa. Cela implique le calcul du CRC, l'assemblage du paquet, et

enfin, la transmission du paquet. Cela accroît la complexité en raison des exigences détaillées de manipulation des paquets.

```
crcResult transmitLoRa(uint16_t *destMac,DataReading *packet,uint8_t
↪  len){}
Logic for constructing and transmitting LoRa packets includes CRC
↪  calculation, packet construction, and transmission
```

Figure 4.9 Code C pour assurer la communication LoRa

De plus, la gestion asynchrone de la transmission et de la réception LoRa introduit de la complexité en gérant les interruptions, les indicateurs et les différents états pour la manipulation simultanée de la transmission et de la réception de données.

L'extrait du code de la fonction 'setFlag', dans la figure 4.10, gère les interruptions et les indicateurs ('enableInterrupt', 'operationDone') pour la communication asynchrone. La complexité provient de la gestion des interruptions et de l'assurance d'états d'indicateurs corrects pour un flux de communication adéquat.

```
volatile bool enableInterrupt=true;
Flag to control interrupt
volatile bool operationDone=false;
Flag indicating packet sent/received
#if defined(ESP8266)||defined(ESP32)
ICACHE_RAM_ATTR
#endif
void setFlag(void)
Handling interrupt by setting operationDone flag
Enable/disable based on the enableInterrupt flag
```

Figure 4.10 Code C pour gérer la communication asynchrone dans LoRa

4.4.0.4 Système IoT en C++

Project-chip/connectedhomeip est un référentiel pour une norme de connectivité de couche applicative unifiée et open-source, conçue pour permettre aux développeurs et aux fabricants

de périphériques de se connecter et de construire des écosystèmes fiables et sécurisés, tout en augmentant la compatibilité entre les appareils domestiques connectés.

Le code examiné du fichier ontentAppPlatform.cpp situé dans (www.github.com/project-chip/connectedhomeip/blob/src/app/app-platform/ContentAppPlatform.cpp#L4) traite des objets connectés dynamiques et de leurs attributs associés.

Dans la figure 4.11, la méthode emberAfExternalAttributeReadCallback gère les opérations de lecture d'attributs, respectivement, pour les objets connectés dynamiques. Dans le même fichier, il y avait une fonction similaire emberAfExternalAttributeWriteCallback, qui effectue des opérations d'écriture. Le code vérifie si l'objet connecté dynamique correspond à une application de contenu connue. S'il est trouvé, il appelle le gestionnaire spécifique de l'application ; sinon, il revient à un gestionnaire générique. Cela démontre la complexité de la gestion de différentes opérations d'attributs basées sur des objets connectés dynamiques et la gestion de scénarios où l'application n'est pas disponible pour un objet connecté donné, ce qui résulte en une base de code complexe.

La gestion du contrôle d'accès des objets connectés introduit de la complexité pour le code des systèmes IoT. La figure 4.12 donne un aperçu du code, qui offre et révoque les permissions pour divers dispositifs. Elle présente une fonction qui gère le contrôle d'accès en créant des entrées ACL.

4.4.0.5 Système IoT en Csharp

Nous étudions **renode/renode**, qui est un framework open-source de simulation et de développement virtuel pour des systèmes IoT embarqués complexes.

Nous présentons une classe nommée **ArduinoLoader** dans l'espace de noms **Antmicro. Renode.Integrations** du fichier ArduinoLoader.cs situé dans (www.github.com/renode/renode/blob/b254f5d2f593e612da80dbb2337fb6394028eca8/src/Renode/Integrations/ArduinoLoader.cs#L27).

```
EmberAfStatus emberAfExternalAttributeReadCallback
(EndpointId endpoint,ClusterId clusterId,const
 ↪  EmberAfAttributeMetadata* attributeMetadata,uint8_t* buffer,
 ↪  uint16_t maxReadLength){}
uint16_t endpointIndex=emberAfGetDynamicIndexFromEndpoint(endpoint);
ChipLogDetail(DeviceLayer,
 ↪  "emberAfExternalAttributeReadCallback endpoint%d",endpointIndex);
EmberAfStatus ret=EMBER_ZCL_STATUS_FAILURE;
ContentApp*app=ContentAppPlatform::GetInstance().GetContentApp(
 ↪  endpoint);
if(app!=nullptr){}
Handle attribute read based on dynamic endpoint
ret=app->HandleReadAttribute(clusterId,attributeMetadata->attributeId,
 ↪  buffer,maxReadLength);
else
If the app is not found for the dynamic endpoint, use a generic
 ↪  handler
ret=AppPlatformExternalAttributeReadCallback(endpoint,rclusterId,
 ↪  attributeMetadata,buffer,maxReadLength);
return ret;
```

Figure 4.11 Code C++ pour déterminer la correspondance entre les points de terminaison dynamiques

La classe configure des objets connectés USB, des configurations et des descripteurs fonctionnels. Elle implique la configuration de plusieurs interfaces USB, objets connectés et descripteurs, ce qui est complexe, comme présenté dans la figure 4.13.

Étant donné qu'il s'agit d'un système IoT, il y a un transfert de données. La méthode Decode traite les données entrantes comme illustré dans la figure 4.14. Elle parcourt les données d'entrée en interprétant les caractères ASCII. Selon le type de caractère, elle ajoute des demi-octets pour former des valeurs numériques. Des déclarations switch gèrent les caractères, indiquant différents types de commandes. Le code prend en charge divers cas, ce qui rend la base de code complexe.

```
// Example of managing access control with ACLs
// and bindings
CHIP_ERROR ContentAppPlatform::ManageClientAccess(Messaging::
 ↪  ExchangeManager&exchangeMgr, SessionHandle&sessionHandle,uint16_t,
 ↪  targetVendorId,uint16_t targetProductId,NodeId localNodeId,std::
 ↪  vector<Binding::Structs::TargetStruct::Type> bindings,Controller::
 ↪  WriteResponseSuccessCallback successCb,Controller::
 ↪  WriteResponseFailureCallback failureCb){}
// Logic for managing ACLs and bindings
// Creation and handling of ACL entries and bindings
// based on vendor and product IDs
 ...
 ...
return CHIP_NO_ERROR;
```

Figure 4.12 Code C++ pour le contrôle d'accès avec ACL et liaisons

```
USBEndpoint interruptEndpoint = null;
// ... (USB configuration)
USBCore = new USBDeviceCore(this,classCode: USBClassCode.
 ↪  CommunicationsCDCControl,maximalPacketSize: PacketSize.Size16,
 ↪  vendorId: 0x2341,productId: 0x805a,deviceReleaseNumber: 0x0100).
 ↪  WithConfiguration(configure: c => c.WithEndpoint(Direction.
 ↪  DeviceToHost, EndpointTransferType.Interrupt, maximumPacketSize:
 ↪  0x08, interval: 0x0a, createdEndpoint: out interruptEndpoint))
// ... (configuring USB interfaces, endpoints, and descriptors)
// ...
```

Figure 4.13 Code Csharp pour la configuration des points de terminaison USB, des configurations et des descriptions fonctionnelles

4.4.0.6 Système IoT en Python

Le projet **DT42/BerryNet** est un système AI/IoT qui connecte des composants indépendants. Les types de composants incluent, mais ne se limitent pas à, un moteur d'IA, un processeur d'E/S, un processeur de données (algorithme) ou un collecteur de données.

```
private void Decode(byte[]d){}
this.Log(LogLevel.Noisy, "Decoding input:{0}", System.Text.
↪  ASCIIEncoding.ASCII.GetString(d));
uint value = 0;
uint savedValue = 0;
var command = Command.None;
for(var i=0;i<d.Length;i++){}
if(d[i]>='0'&&d[i]<='9'){}
AppendNibble(ref value,(byte)(d[i]-'0'));
else if(d[i] >= 'a' && d[i] <= 'f'){}
AppendNibble(ref value,(byte)(d[i]-'a'));
else if(d[i]>='A'&&d[i]<='F'){}
AppendNibble(ref value,(byte)(d[i]-'A'));
else{}
switch((char)d[i]){}
// ... (handling various cases)
}}}}
```

Figure 4.14 Code Csharp pour l'encodage et le décodage des commandes entrantes

Le fichier bnpipeline.py (www.github.com/DT42/BerryNet/blob/berrynet/bndyda/bnpipeline.py#L4) définit des classes liées à un moteur de pipeline pour le traitement de données dans un contexte AI/IoT.

La complexité de la base de code étudiée découle de son comportement dynamique, de ses nombreuses options de configuration, de la gestion de la communication et de la nécessité de gérer différents modes et moteurs en fonction des messages externes. Bien que ces fonctionnalités offrent une flexibilité à l'IoT, elles augmentent également la complexité globale de la base de code.

Dans la figure 4.15, nous présentons du code assurant un commutateur dynamique entre un moteur de pipeline réel (PipelineEngine) et un moteur fictif (PipelineDummyEngine) en fonction des messages MQTT indiquant le mode de service (inférence, inactif ou apprentissage). Ce changement dynamique ajoute de la complexité au code.

```
if mode=='inference':self.disable_engine=False
self.engine=PipelineEngine(...)
else:
self.disable_engine=True
self.engine=PipelineDummyEngine()
```

Figure 4.15 Code Python pour assurer la commutation dynamique du moteur

Le code assure la communication avec un courtier MQTT, gérant divers sujets et messages. Cela comprend l'envoi de résultats, le déploiement de nouveaux modèles réentraînés et le passage entre les modes d'inférence et de non-inférence. L'utilisation de MQTT pour la communication introduit de la complexité dans la base de code. La figure 4.16 est un exemple de gestion de la communication reflètant cette complexité.

```
self.comm.send('berrynet/engine/pipeline/result', tools.dump_json(
↪  generalized_result))
```

Figure 4.16 Code Python pour assurer la communication avec un courtier MQTT

4.5 Implications pratiques pour le développement IoT

Nous présentons une discussion sur les implications des valeurs observées des métriques de qualité mesurées et de notre analyse approfondie, ainsi que de leurs résultats sur le développement pratique d'IoT. Nous relions ces observations à des défis du monde réel et proposons des implications pour les développeurs et les praticiens IoT.

Défi du monde réel et observation 1 : Le développement IoT implique des interactions complexes entre le matériel et le logiciel, des communications de données complexes et des algorithmes complexes. Des métriques telles que LOC, #Classes, #Fichiers, CC, HV, et WMC mettent en évidence la nature extensive et complexe du code dans les systèmes IoT. L'analyse de code approfondi dans la section 4.4 illustre également la difficulté de comprendre le code des systèmes IoT.

Implication 1 : Les développeurs engagés dans des projets IoT devraient cultiver des compétences spécialisées, telles qu'une compréhension approfondie des aspects matériels et logiciels et une expertise dans des protocoles de communication de données efficaces pour surmonter les défis posés par les interactions matériel-logiciel, les intricacités de la communication de données et les algorithmes complexes.

Défi du monde réel et observation 2 : Une interdépendance plus élevée entre différents modules au sein d'un système et une faible maintenabilité dans les systèmes IoT, reflétées dans des métriques telles que RFC, CBO, et MI, posent des défis pour apporter des modifications et maintenir la base de code.

Implication 2 : Mettre l'accent sur la conception modulaire et une organisation efficace du code permet d'encapsuler la fonctionnalité en unités distinctes et gérables, essentielles pour gérer efficacement l'interdépendance, la maintenabilité et les bases de code étendues dans les systèmes IoT.

Défi du monde réel et observation 3 : Le nombre accru de classes et de fichiers dans les systèmes IoT est induit par leur nature distribuée. Les calculs statistiques sur les métriques révèlent que les systèmes IoT présentent un code plus étendu par rapport aux systèmes non IoT. Les métriques montrent une différence notable dans le nombre de classes et de fichiers, comme présenté dans le Tableau 4.5.

Implication 3 : Les développeurs engagés dans des projets IoT devraient privilégier la mise en œuvre de pratiques organisationnelles et structurelles efficaces, telles que l'élimination de segments de code redondants et l'optimisation des bibliothèques de fonctions. Ces pratiques impliquent la réduction de la taille du code.

Défi du monde réel et observation 4 : Comprendre et maintenir la qualité du code dans des projets IoT évolutifs. Comprendre le compromis entre des métriques telles que RFC, CBO, LCOM, CC et WMC dans les systèmes IoT offre aux développeurs des informations exploitables

pour prendre des décisions éclairées et des actions spécifiques dans le développement afin d'améliorer les valeurs des métriques.

Implication 4 : La surveillance continue des métriques de code pour adapter le développement du système IoT en fonction des résultats de ces métriques, est essentielle pour garantir sa qualité.

4.6 Résultats sur les bonnes pratiques

Notre comparison révèle des différences entre les systèmes IoT et non IoT. Par conséquent, nous utiliserons ces informations pour proposer des bonnes pratiques tirées de la littérature non IoT afin de fournir des directives spécifiques pour relever les défis spécifiques à l'IoT découverts grâce à notre comparaison, y compris le couplage élevé, la faible cohésion, la complexité élevée, la faible maintenabilité, la réduction de la taille du code et la lisibilité.

Nous présentons les bonnes pratiques par catégorie, comme indiqué dans le Tableau 4.8. Certaines des bonnes pratiques identifiées, telles que la modularité et le refactoring, peuvent résoudre plusieurs problèmes, c'est pourquoi nous les retrouvons mentionnées sous différentes catégories. Une version plus détaillée du Tableau 4.8, incluant des informations supplémentaires, est disponible dans notre paquet de réplications accessible sur Zenodo (www.zenodo.org/records/10564976).

4.6.1 Taille

Nous observons que la taille du code est plus importante dans les systèmes IoT, comme indiqué par des valeurs élevées de la métrique mesurée LOC et le nombre accru de classes et de fichiers dans les systèmes IoT. L'exécution de la requête a produit 530 000 articles parmi lesquels nous sélectionnons les dix articles les plus cités. Pour résoudre les problèmes de taille précédemment démontrés dans les sections 4.2, 4.3, et 4.4, nous avons identifié ces bonnes pratiques.

Tableau 4.8 Bonnes pratiques pour diverses catégories de métrique

Catégorie	Les bonnes pratiques	Outils	Techniques	Disponibilité	Applicabilité	Raison
Taille	Techniques d'optimisation du code	Cppcheck	Éliminer le code en double, identifier les opportunités d'optimiser des boucles, et réduire les opérations coûteuses	Oui	Oui	Cppcheck permet l'optimisation du code.
	Identification et consolidation des fonctions similaires	NA	NA	Non	Non	Des algorithmes sont disponibles, mais aucun logiciel.
	Utilisation de la décompression à l'exécution	IBM CodePack	NA	Non	Oui	IBM's CodePack n'est pas open source.
Complexité	Application de refactoring	Eclipse, SonarQube, Checkstyle	Remonter, extraire et enliner des méthodes	Oui	Oui	Les outils sont disponibles et peuvent être utilisés.
	Application de la modularité	Docker, Virtual Box	Principes d'encapsulation et d'abstraction	Oui	Oui	Les outils sont disponibles
	Utilisation de composants logiciels empaquetés	ThingSpeak, Microsoft Azure IoT Suite, Google Cloud IoT	NA	Non	Oui	Les outils disponibles ne sont pas open source.
Couplage et Cohésion	Application des principes et des patrons de conception	NA	Patron DI pour IoC, Responsabilité Unique et Singleton	Oui	Oui	Plusieurs études ont montré que les patrons de conception peuvent être appliqués aux systèmes IoT.
	Application de refactoring	Eclipse, SonarQube, Checkstyle	NA	Oui	Oui	Le refactoring pour l'IoT est possible avec Eclipse par exemple.
	Application de la modularité	Docker, Virtual Box	Techniques d'analyse de cluster	Oui	Oui	Des outils de conteneurisation sont disponibles, comme Docker.
Lisibilité du code	Utilisation des caractéristiques textuelles	ESLint, Pylint, ou Checkstyle	Utilisation de lignes de code plus courtes et indentation, commentaires, lignes vides, variables significatives et descriptives, etc.	Oui	Oui	Les techniques s'implimentent au phase développement du système.
	Amélioration de l'entropie du code	NA	Améliorer l'organisation générale, la structure et la variabilité du code	Oui	Oui	Les développeurs doivent surveiller la valeur de l'entropie lors du développement du système.
Maintenabilité	Utilisation des conventions de code source	FindBugs1, Checkstyle2 et Jtest3	NA	Oui	Oui	Cette utilisation est dans la phase de développement du système.
	Utilisation de l'architecture pilotée par modèle	ThingML, Papyrus	NA	Oui	Oui	Les techniques disponibles s'appliques à la conception du système.
	Utilisation de patrons de conception	Eclipse	Patron Factory, Singleton et Decorator	Oui	Oui	Peuvent être appliquées à la phase de conception du système.
	Application de refactoring	Eclipse, SonarQube, Checkstyle	Encapsulation, limitation de la longueur des unités de code à 15 lignes de code, limitation du nombre de points de branchement par unité à 4, etc.	Oui	Oui	La refactoring pour l'IoT est possible.
	Intégration Continue et Déploiement Continue (CI/CD)	Application Azure IoT Edge, CircleCI, Jenkins comme gestionnaire CI/CD IoT	NA	Oui	Oui	Technique a intégrer aux systèmes au phase de développement.

4.6.1.1 Techniques d'optimisation du code

Plusieurs études ont introduit des techniques pour réduire la taille du code (Hanson, 1983; Edler von Koch, Franke, Bhandarkar & Dasgupta, 2014). La plupart de ces techniques peuvent être utilisées dans les systèmes IoT, telles que le déroulement des boucles, la réduction de la complexité en remplaçant des opérations coûteuses par des alternatives moins gourmandes en ressources, l'inlining de fonctions pour minimiser les appels de fonction, la réduction de la complexité des tableaux, l'élimination des calculs redondants, la suppression de code dupliqué, et l'optimisation des bibliothèques de fonctions en sélectionnant des dépendances légères et en supprimant le code inutilisé.

Les outils d'analyse statique peuvent aider à mettre en œuvre ces techniques, tels que Cppcheck, utilisé dans les systèmes embarqués et le développement IoT pour le code C et C++ Cppcheck (2021). Cppcheck peut également détecter des opportunités d'optimisation de boucles ou suggérer de bonnes façons de gérer les itérations, ayant ainsi un impact indirect sur la taille du code en réduisant le nombre d'instructions exécutées.

4.6.1.2 Identification et consolidation des fonctions similaires

La réduction de la taille du code est possible grâce à l'identification et à la consolidation de fonctions similaires.

L'un des articles sélectionnés est le travail d'Edler von Koch *et al.* (2014), qui propose une technique d'optimisation de code indépendante de la platforme pour réduire la taille du code en fusionnant des fonctions structurellement similaires. L'algorithme de fusion de fonctions compare les signatures de fonctions et les graphes de flux de contrôle pour détecter les équivalences.

La nature indépendante de la platforme de l'algorithme, fonctionnant au niveau de la représentation intermédiaire dans une machine virtuelle basse (LLVM), le rend adaptable à la diversité des dispositifs IoT avec des architectures variables en abstrayant les détails spécifiques du matériel et en permettant la génération de code adapté à différents environnements cibles. Les paramètres de

l'algorithme, y compris le nombre minimum d'instructions et les seuils de similarité, contribuent à son adaptabilité, garantissant que le processus de fusion réponde aux contraintes spécifiques des environnements IoT.

4.6.1.3 Utilisation de la décompression à l'exécution

Les techniques de décompression à l'exécution permettent une réduction de la taille du code. Cette décompression à l'exécution implique l'utilisation de techniques telles que la décompression logicielle basée sur un dictionnaire et la compression sélective. Lefurgy, Piccininni & Mudge (2000) ont proposé une décompression logicielle basée sur un dictionnaire, un décompresseur logiciel basé sur IBM CodePack, et une technique de compression sélective pour contrôler la dégradation des performances due à la décompression, en utilisant des caches gérés par logiciel pour prendre en charge la décompression du code à la granularité d'une ligne de cache.

Les techniques que nous pouvons adapter pour le développement IoT à partir de la décompression à l'exécution comprennent la décompression sélective, la décompression dynamiques adaptés à la disponibilité des ressources et la décompression conditionnelle. Cependant, la décompression de programmes entiers, la décompression en temps réel de grands corpus de code et les techniques de compression de données peuvent être moins pratiques pour de nombreux dispositifs IoT et composants avec une mémoire et une puissance de traitement limitées.

4.6.2 Complexité

Nous observons une complexité élevée dans les systèmes IoT, comme indiqué par des valeurs élevées de métriques mesurées telles que CC et WMC. La requête a abouti à 366 000 articles, et nous examinons les dix articles les plus cités. Pour résoudre les problèmes de complexité précédemment démontrés dans les sections 4.2, 4.3, et 4.4, nous identifions ces bonnes pratiques.

4.6.2.1 Application de refactoring

Les méthodes de refactoring (Fowler, 2018) offrent une série de stratégies pour réduire la complexité du code (Soetens & Demeyer, 2010).

Le refactoring implique la redistribution des variables et des méthodes à travers la hiérarchie des classes pour simplifier la structure du système logiciel, avec des techniques telles que le regroupement, l'extraction et l'inlining des méthodes. Mayer (2022) souligne l'importance régulière de refactoring de code dans le développement pour décomposer les fonctions complexes. Bien que le refactoring s'applique aux systèmes IoT, elle peut introduire des bogues de concurrence et des changements de comportement (Zhang, Sun, Zhang, Qiu & Tian, 2020b), nécessitant une détection et une évaluation post-refactoring pour les corrections.

Pour les systèmes IoT basés sur Java, le processus de refactoring peut être exécuté de manière transparente avec Eclipse IDE, en utilisant ses outils de refactoring intégrés. Des outils d'analyse statique du code comme SonarQube ou Checkstyle peuvent identifier des zones potentielles pour le refactoring. Leur intégration dans le pipeline de développement pour une analyse statique continue du code et des suggestions d'amélioration contribue à réduire la complexité du code.

4.6.2.2 Application de la modularité

La modularité du code est une technique importante pour réduire la complexité, soulignée dans la théorie de la modularité de Baldwin & Clark (2000) et dans le travail de Kearney, Sedlmeyer, Thompson, Gray & Adler (1986). Cette technique met en avant les avantages de la décomposition de systèmes complexes en modules plus petits et plus gérables. Ce principe de modularité peut être appliqué aux systèmes IoT en décomposant un système IoT en composants modulaires. Des outils de conteneurisation comme Docker sont disponibles pour faciliter les principes d'encapsulation et d'abstraction, qui sont des techniques qui contribuent à une meilleure modularité du code. Docker utilise la conteneurisation pour encapsuler les applications et leurs dépendances, créant ainsi des environnements isolés. Dans les systèmes IoT, nous pouvons créer

des conteneurs Docker pour différents composants ou services et empaqueter chaque composant avec ses dépendances dans une image Docker distincte.

4.6.2.3 Utilisation de composants logiciels empaquetés

Les composants logiciels empaquetés sont des modules logiciels préconstruits et prêts à l'emploi qui peuvent être intégrés dans un système logiciel plus vaste. Leur utilisation est associée à une complexité logicielle réduite Banker, Davis & Slaughter (1998).

Dans le développement IoT, où ce concept, comme les plateformes IoT, est courant, ces conclusions revêtent une importance particulière. Des exemples de composants logiciels empaquetés comprennent les plateformes IoT, qui offrent des outils et des services pour la construction et la gestion d'applications IoT à l'aide d'outils tels que ThingSpeak, Microsoft Azure IoT Suite, Google Cloud IoT et IBM Watson IoT Platform.

4.6.3 Couplage et cohésion

Les systèmes IoT présentent un couplage élevé (RFC et CBO élevés) et une faible cohésion (LCOM faible) par rapport aux systèmes non IoT. Nous avons trouvé 26 400 articles et sélectionné les dix meilleurs en fonction des citations. À partir de ces dix articles, nous extrayons les bonnes pratiques pour résoudre les problèmes de couplage et de cohésion précédemment démontrés dans les sections 4.2, 4.3, et 4.4.

4.6.3.1 Application des principes et des patrons de conception

Walls & Breidenbach (2007) ont montré que l'injection de dépendances (DI) permet d'obtenir une inversion de contrôle (IoC), conduisant à un couplage réduit et à une amélioration de la cohésion du code.

Dans les environnements IoT contraints en ressources, nous n'avons pas la possibilité d'utiliser des DI complets. Cependant, nous pensons que l'utilisation de DI légers à travers une bibliothèque

DI légère telle que TinyIoC ou MicroDI, conçue pour les systèmes embarqués et IoT, est utile. Ces DI fournissent une fonctionnalité de base sans le surcoût.

Les patrons Singleton et Factory (Alrubaye, Alshoaibi, Alomar, Mkaouer & Ouni, 2020; AlOmar, AlRubaye, Mkaouer, Ouni & Kessentini, 2021) garantissent des responsabilités individuelles de classe, améliorant la cohésion et réduisant le couplage. Pour l'IoT, l'application de ces modèles est simple et facilite le découplage des modules, simplifiant la séparation des rôles et atténuant l'hétérogénéité des dispositifs (Jung, Cho & Kang, 2014). Lors de l'application de ces patrons, il y a des différences minimes par rapport aux contextes non IoT qui doivent être prises en compte, notamment l'optimisation des implémentations de modèles de conception personnalisées pour les systèmes IoT fonctionnant dans des environnements contraints en ressources.

4.6.3.2 Application de refactoring

Tout comme pour la complexité, Du Bois, Demeyer & Verelst (2004) ont proposé un guide de refactoring pour améliorer le couplage et la cohésion du code. Il est crucial d'organiser le code avec des fonctionnalités liées regroupées et de séparer différentes préoccupations dans des modules ou des classes distincts (Alrubaye *et al.*, 2020). Le refactoring améliore le couplage en réduisant les interconnexions entre les modules en minimisant les appels de méthode et les variables partagées Alrubaye *et al.* (2020).

Nous constatons que ces principes de refactoring s'appliquent aux systèmes IoT sur la base de notre étude des étapes de refactoring (Du Bois *et al.*, 2004). Il existe des outils et des fonctionnalités IDE disponibles pour que les développeurs puissent identifier automatiquement et suggérer des refactorings pour les systèmes IoT (AlOmar *et al.*, 2021), tels qu'Eclipse, SonarQube et Checkstyle.

4.6.3.3 Application de la modularité

La modularité est une bonne pratique bien connue pour améliorer le couplage et la cohésion. L'utilisation de techniques d'analyse de cluster peut évaluer et améliorer la modularisation

(e Abreu & Goulao, 2001). Dans le contexte IoT, la catégorisation sémantique peut être utilisée pour regrouper les composants IoT en fonction de leurs rôles (par exemple, capteurs, actionneurs, contrôleurs), et la combinaison de critères structurels et sémantiques améliore de manière approfondie la modularisation.

Pour améliorer la modularité, des techniques d'analyse de cluster peuvent être appliquées (e Abreu & Goulao, 2001). Dans le système IoT, cela implique d'analyser les relations et les dépendances entre différents composants ou modules. En identifiant les interrelations entre divers dispositifs ou composants IoT, nous pouvons créer des modules plus cohésifs et moins couplés.

4.6.4 Lisibilité du code

Les systèmes IoT présentent une lisibilité de code supérieure, comme indiqué par leurs valeurs CP plus élevées par rapport aux systèmes non IoT. La requête de recherche a produit 66 700 articles, et nous avons choisi les dix articles les plus cités.

Une difficulté significative dans les études de lisibilité réside dans la complexité de justifier expérimentalement ce qui constitue essentiellement une perception subjective. Obtenir des mesures de perception subjective est difficile, nécessitant des études humaines et impliquant intrinsèquement une variabilité. Pour obtenir des mesures utiles, des enquêtes à grande échelle comprenant de multiples évaluateurs humains et une analyse statistique minutieuse de l'accord inter-évaluateurs sont essentiels (Posnett, Hindle & Devanbu, 2011). Nous rapportons les bonnes pratiques identifiées comme étant utiles pour améliorer la lisibilité du code.

4.6.4.1 Utilisation de caractéristiques textuelles

L'utilisation de caractéristiques textuelles simples améliore la lisibilité du code, en mettant l'accent sur l'importance des lignes courtes, de l'indentation cohérente et de l'utilisation judicieuse des commentaires (Buse & Weimer, 2009; Piantadosi, Fierro, Scalabrino, Serebrenik & Oliveto, 2020). Bien que les commentaires ne reflètent pas uniformément une lisibilité

élevée, ils communiquent directement l'intention, ce qui les rend préférables. Les lignes vides, positivement corrélées à la lisibilité (Buse & Weimer, 2009; Sampaio & Barbosa, 2016).

Wang, Pollock & Vijay-Shanker (2011) proposent SEGMENT une solution heuristique d'insertion automatique de lignes vides basée sur la structure du programme et les informations de dénomination. Adapter les heuristiques de SEGMENT au code IoT en tenant compte d'éléments structurels tels que les gestionnaires d'événements, le traitement des données et les tâches de communication permet d'insérer des lignes vides entre des segments de code logiquement liés, améliorant ainsi la lisibilité.

Des noms de variables significatifs et des noms de méthode descriptifs sont importants pour la lisibilité du code (Sampaio & Barbosa, 2016; Sedano, 2016). En développement IoT, l'utilisation de noms clairs et descriptifs pour les variables représentant les capteurs, les actionneurs et les données améliore la lisibilité du code, en particulier lorsque les méthodes interagissent avec les capteurs ou effectuent des tâches spécifiques.

Pour mettre en œuvre ces techniques, des examens manuels du code axés sur les fonctionnalités textuelles mentionnées ou le développement de scripts personnalisés adaptés aux langages de programmation IoT sont des options viables. Alternativement, des outils existants d'analyse statique du code prenant en charge les métriques de lisibilité, telles que ESLint, Pylint ou Checkstyle, peuvent être adaptés ou étendus pour répondre aux exigences mentionnées.

4.6.4.2 Améliorer l'entropie du code

Posnett *et al.* (2011) suggèrent que les extraits de code avec une entropie plus élevée sont plus lisibles. Le concept d'entropie mesure la quantité de contenu informationnel dans le code source. Il est souvent considéré comme la complexité, le degré de désordre ou la quantité d'informations dans un signal ou un ensemble de données. L'entropie est calculée à partir du nombre de termes (tokens ou octets) ainsi que du nombre de termes et d'octets uniques.

Le code avec des éléments plus variés (opérateurs et opérandes) est plus facile à comprendre. Lors de la programmation, les développeurs doivent améliorer la variabilité du code.

En IoT, nous pouvons traiter une variété de capteurs, d'actionneurs et de protocoles de communication. Les développeurs, lors de la création de systèmes IoT, doivent surveiller de manière constante la valeur de l'entropie à travers divers éléments (opérateurs et opérandes) dans le code pour améliorer son entropie globale à l'aide d'outils d'analyse statique du code.

4.6.5 Maintenabilité

La maintenabilité des systèmes IoT est faible par rapport aux systèmes non IoT ; cela est prouvé par la faible valeur de MI, la complexité élevée du code et la forte interdépendance entre différents modules au sein d'un système. La recherche a abouti à 55 300 articles, et nous avons examiné les dix articles les plus cités. Pour résoudre les problèmes de maintenabilité précédemment identifiés dans les sections 4.2, 4.3, et 4.4, nous avons trouvé les bonnes pratiques suivantes.

4.6.5.1 Utilisation des conventions et normes de code source

Les conventions de code source et les langages de programmation ont évolué ensemble, en respectant des conventions uniformes telles que les conventions de nommage, la documentation en ligne et la structure syntaxique, ce qui améliore la lisibilité du code. Gergel, Stroulia, Smit & Hoover (2011) ont énuméré des conventions de code cruciales pour la maintenabilité, particulièrement pertinentes pour Java. Ces conventions incluent des recommandations pour les déclarations If, For et Try, suggérant au plus une déclaration imbriquée supplémentaire, préconisant la conception de classes extensibles sans code dans les méthodes publiques, et plus encore.

Cette liste de conventions est largement applicable au code des systèmes IoT. La mise en œuvre de ces conventions peut être facilitée en utilisant des outils tels que FindBugs, Checkstyle et Jtest.

4.6.5.2 Utilisation de l'architecture pilotée par les modèles (MDA)

La MDA consiste à exprimer les exigences du système dans un langage de modélisation (par exemple, UML) pour générer un modèle indépendant de la plateforme (PIM). Ce PIM est ensuite transformé en un modèle spécifique à la plateforme (PSM) pour une technologie particulière, puis en code réel. La MDA améliore la maintenance du système en facilitant les modifications au niveau des exigences, les propageant automatiquement aux modules concernés (da Silva, Maciel & Ramalho, 2013; Moadad, Damaj & El Kabani, 2022).

Dans le développement IoT, la MDA peut être exploitée pour créer des modèles de haut niveau capturant les exigences du système et des spécificités telles que l'intégration des capteurs, le traitement des données et les protocoles de communication. L'application de la MDA dans l'IoT garantit une génération de code basée sur ces modèles, améliorant la maintenabilité du code et réduisant les erreurs Bowles (2004).

4.6.5.3 Utilisation des patrons de conception

En plus du couplage et de la cohésion, les patrons de conception impactent positivement la maintenabilité du code (Hegedus, 2013). Jun & Rana (2021) ont démontré de manière empirique que l'utilisation efficace de patrons de conception améliore la maintenabilité du logiciel grâce à l'évaluation d'un système sans patrons de conception par rapport à sa version affinée après l'application de patrons de conception appropriés.

L'utilisation de patrons de conception dans les systèmes IoT est directe. Par exemple, le patron de conception Factory Method facilite la création d'objets sans spécifier de classes concrètes, facilitant l'intégration de nouveaux types de dispositifs ou de fonctionnalités dans un contexte IoT. Le pattern Decorator permet l'ajout dynamique de responsabilités aux objets, permettant l'amélioration flexible des capacités des dispositifs IoT sans altérer leur structure de base. Des outils tels qu'Eclipse pour les systèmes Java peuvent aider à la mise en œuvre de ces patrons.

4.6.5.4 Application de refactoring

Similairement à la complexité, au couplage et à la cohésion, les techniques de refactoring impactent positivement la maintenabilité du logiciel (Hegedus, 2013) et réduisent la dette technique (Samoladas, Stamelos, Angelis & Oikonomou, 2004).

Pour le code C#, Visser, Rigal, Wijnholds, Van Eck & van der Leek (2016) ont fourni des directives pour l'amélioration de la maintenabilité par le refactoring. Cela inclut la limitation de la longueur des unités de code (méthodes ou constructeurs) à 15 lignes, la restriction du nombre de points de branchement par unité à 4 (diviser les unités complexes en unités plus simples) et l'équilibrage de la taille relative des composants de niveau supérieur.

Le refactoring du code dans les solutions IoT nécessite une compréhension de l'architecture du système et de ses implications sur le flux de données et les protocoles de communication, facilitant la restructuration du code pour une maintenabilité améliorée sans altérer le comportement externe des systèmes IoT. Tout en refactorant, nous pouvons mettre en œuvre l'encapsulation, qui, comme le préconisent Anda (2007), améliore la maintenabilité en masquant les détails du système. En IoT, l'encapsulation implique de cacher les détails internes des dispositifs IoT et de leurs protocoles de communication.

4.6.5.5 Intégration continue et déploiement continue (CI/CD)

La mise en œuvre de pipelines CI/CD pour automatiser les processus de test et de déploiement améliore la maintenabilité (Samoladas *et al.*, 2004; Visser *et al.*, 2016).

Pour les systèmes IoT, la CI/CD facilite les mises à jour rapides et fiables des dispositifs IoT. Cependant, il y a des considérations spécifiques à prendre en compte avant de l'appliquer à l'IoT, comme la création de simulations réalistes de dispositifs IoT pour les tests. Des outils et des frameworks tels qu'Eclipse Kapua, IoTivity et IoT-LAB peuvent simuler le comportement et les interactions des dispositifs IoT. Les mécanismes de mise à jour sont essentiels pour déployer à distance des mises à jour du micrologiciel sur les dispositifs IoT. L'ensemble du pipeline CI/CD

peut être réalisé via l'application Azure IoT Edge, CircleCI et Jenkins en tant que gestionnaire CI/CD IoT.

4.7 Discussion

Nos résultats révèlent que la qualité du code des systèmes IoT est inférieure à celle des systèmes non IoT, ce qui impacte la qualité globale du code avec des valeurs de métriques plus élevées : WMC, RFC, CC, CBO et LOC. Cela suggère que les systèmes IoT complexes, avec plusieurs couches de matériel, de logiciel et de connectivité, posent des défis pour maintenir un code de haute qualité (Taivalsaari & Mikkonen, 2017). Les systèmes IoT fonctionnent dans des environnements aux ressources limitées, et des algorithmes complexes sont souvent nécessaires pour des matériels spécifiques, rendant le code plus difficile à comprendre, ce qui peut affecter sa qualité. Les environnements à ressources limitées compliquent la détection d'erreurs et le débogage, car les ressources limitées restreignent les tests et le débogage. Cela se traduit par une qualité du code inférieure car les développeurs effectuent moins de tests et mettent en œuvre des pratiques de gestion des erreurs limitées.

IoT introduit un concept distinct par rapport aux non IoT, à savoir la connectivité. Les appareils IoT communiquent avec d'autres appareils et le cloud, ce qui peut entraîner une communication peu fiable ou intermittente. Nous pensons que la complexité des systèmes IoT est affectée par ces défis de connectivité (Bures *et al.*, 2021), par la mise en œuvre de protocoles de communication et de mécanismes de réessaie.

4.8 Menaces pour la validité

Nos résultats sont soumis à une menace pour leur validité interne, externe et des conclusions. Nous énumérons ici ces menaces et ce que nous avons fait pour les minimiser.

Validité interne :

La méthodologie proposée pour la comparaison entre les systèmes IoT et non IoT est structurée et systématique, débutant par la sélection de dépôts sur GitHub basée sur des critères tels que les nombres d'étoiles, les *forks* et les étiquettes. Cette première étape est appropriée pour identifier des dépôts représentatifs dans chaque catégorie et à été utiliser dans des travaux précédents (Politowski *et al.*, 2021) et (Corno *et al.*, 2020). Cependant, la concentration sur les métriques de popularité comme les étoiles et les forks pourrait introduire un biais, car la popularité ne garantit pas nécessairement la qualité ou la pertinence du système. En outre, la sélection manuelle des dépôts pourrait introduire un autre biais, car elle dépend de l'interprétation de code subjective des chercheurs. Enfin, bien que la méthodologie prévoie une comparaison des métriques de code, elle ne prend pas en compte de facteurs externes tels que les objectifs du projet ou les exigences spécifiques du domaine, ce qui pourrait limiter la portée et la pertinence des conclusions tirées de la comparaison.

L'utilisation d'un ensemble limité de mesures de qualité pourrait ne pas fournir une représentation complète des systèmes logiciels. Pour remédier à cette limitation, nous avons sélectionné une gamme diversifiée de métriques couramment utilisées provenant de différentes catégories, assurant ainsi une perspective plus holistique de l'analyse statique du code.

Le choix des outils pour mesurer les métriques de qualité peut ne pas correspondre parfaitement aux caractéristiques spécifiques des systèmes IoT et non IoT. Pour atténuer cette préoccupation, nous avons utilisé deux outils d'analyse populaires au lieu de nous appuyer sur un seul outil, améliorant ainsi la précision de nos résultats.

Notre étude se concentre sur une grande variété de projets hétérogènes tels que des bibliothèques de programmation, des frameworks, des bases de données, des IDE, des jeux, des programmes scientifiques, etc. Nous reconnaissons que notre analyse des caractéristiques de ces projets non IoT est limitée dans le cadre de ce travail, ce qui pourrait introduire des biais ou des limitations en raison des différences inhérentes entre ces types de projets. Des recherches futures devraient

utiliser davantage de critères de sélection pour améliorer la complétude et la robustesse de telles analyses.

Validité externe :

Les disparités dans les niveaux d'expérience des développeurs travaillant sur des systèmes IoT et non IoT peuvent influencer les différences de qualité logicielle. Pour remédier à ce biais potentiel, nous avons effectué des analyses manuelles pour assurer la qualité des systèmes sélectionnés. De plus, nous avons identifié et traité les valeurs aberrantes et les anomalies qui pourraient affecter la validité de nos résultats.

Validité des conclusions :

Comparer les systèmes IoT et non IoT est une tâche complexe, car la distinction entre leur code et leurs systèmes est nuancée et complexe. Malgré ces défis, notre travail représente une première étape dans cette analyse comparative, posant les bases pour des recherches futures dans ce domaine.

4.9 Conclusion

Dans ce chapitre, notre objectif était de combler le manque dans la recherche sur la qualité logicielle des systèmes IoT en effectuant une analyse comparative avec des systèmes logiciels non IoT, reconnaissant les défis uniques posés par les ressources limitées et les architectures distribuées de l'IoT. Les résultats mettent en évidence des différences clés dans des métriques telles que la complexité, la cohésion, la taille du code et la maintenabilité, indiquant que le développement de systèmes IoT nécessite des bonnes pratiques adaptées.

Face à ces différences, nous avons compilé systématiquement un ensemble de bonnes pratiques couramment utilisées dans les systèmes non IoT et personnalisé une liste de bonnes pratiques spécifiquement conçues pour le développement de systèmes IoT afin de prendre en compte ces distinctions.

Nous avons sélectionné et analysé systématiquement 94 systèmes IoT et non IoT comparables, fournissant des aperçus complets de leurs bases de code respectives. Nos contributions comprennent une méthode de choix de systèmes comparables, le calcul et l'analyse de diverses métriques, une analyse approfondie du code de certains systèmes IoT, et une liste revisitée des bonnes pratiques pour le développement IoT, abordant des défis observés tels que la complexité élevée, la maintenabilité faible et des problèmes de lisibilité.

CONCLUSION ET TRAVAUX FUTURS

Les systèmes IoT offrent des avantages majeurs en termes d'automatisation et d'efficacité, améliorant la vie quotidienne des individus et permettant aux entreprises d'optimiser leurs processus tout en réduisant les coûts de main-d'œuvre. Cependant, comme tout logiciel, maintenir le bon fonctionnement d'un système IoT nécessite une concentration sur sa qualité logicielle.

Notre recherche approfondie sur la qualité des systèmes IoT, en se penchant sur l'architecture et le code, a révélé des informations cruciales pour les praticiens et les chercheurs. Les systèmes IoT, bien qu'apportant des avantages significatifs, sont confrontés à des défis considérables en matière de qualité logicielle.

L'étude des styles architecturaux a permis de distinguer les forces et les faiblesses de différents styles. Les résultats offrent des recommandations pratiques aux praticiens pour choisir l'architecture la mieux adaptée à leurs besoins spécifiques et aux exigences de qualité. L'identification des architectures les plus utilisées, avec une évaluation approfondie de leurs avantages et inconvénients, fournit des recommandations essentielles pour le choix des architectures de systèmes IoT qui répondent à leurs exigences de qualité.

En ce qui concerne la qualité du code, la comparaison approfondie entre les systèmes IoT et non IoT a mis en lumière des différences significatives. Les systèmes IoT présentent une complexité accrue, un couplage élevé, une largeur importante et une maintenabilité moindre. Ces observations ont conduit à une liste revisitée de bonnes pratiques, offrant des recommandations pour améliorer la qualité du code dans le contexte des systèmes IoT.

Notre travail de recherche est pertinent pour la communauté des chercheurs et des développeurs travaillant dans le domaine de l'IoT. Nous avons fourni des informations essentielles sur les architectures des systèmes IoT, aidant ainsi à la sélection d'une architecture adéquate pour garantir la qualité du système. De plus, nous avons démontré que la qualité du code est un facteur clé pour le succès du développement d'applications IoT et nous avons proposé des pratiques recommandées pour les implémentations futures.

Dans le cadre de travaux futurs, nous pouvons approfondir la recherche sur la qualité des architectures IoT en explorant des modèles architecturaux innovants et en évaluant leur impact sur la qualité des systèmes IoT. Cela pourrait inclure l'investigation de nouvelles approches architecturales émergentes et l'adaptation de celles-ci pour répondre aux exigences spécifiques de différents scénarios IoT. Nous voulons également effectuer une analyse à un niveau de granularité plus élevé que l'analyse du code en examinant et comparant la conception architecturale des systèmes IoT et non IoT.

Côté amélioration de la qualité du code des systèmes IoT, les recherches futures peuvent élaborer des méthodologies de développement spécifiquement axées sur la qualité des systèmes IoT. Cela pourrait impliquer la création de lignes directrices et de frameworks pour faciliter le processus de développement, en mettant l'accent sur l'intégration des meilleures pratiques pour assurer la qualité à toutes les étapes du cycle de vie du développement.

BIBLIOGRAPHIE

Abbade, L. R., da Cruz, M. A., Rodrigues, J. J., Lorenz, P., Rabelo, R. A. & Al-Muhtadi, J. (2020). Performance comparison of programming languages for Internet of Things middleware. *Transactions on Emerging Telecommunications Technologies*, 31(12), e3891.

Achir, M., Abdelli, A., Mokdad, L. & Benothman, J. (2022). Service discovery and selection in IoT : A survey and a taxonomy. *Journal of Network and Computer Applications*, 200, 103331.

Ahmad, A., Fahmideh, M., Altamimi, A. B., Katib, I., Albeshri, A., Alreshidi, A., Alanazi, A. A. & Mehmood, R. (2021). Software Engineering for IoT-driven data analytics applications. *IEEE Access*, 9, 48197–48217.

Ahmad, N. & Zulkifli, A. M. (2022). Internet of Things (IoT) and the road to happiness. *Digital Transformation and Society*, 1(1), 66–94.

Ahmed, A. I. A., Gani, A., Ab Hamid, S. H., Abdelmaboud, A., Syed, H. J., Mohamed, R. A. A. H. & Ali, I. (2019a). Service management for IoT : requirements, taxonomy, recent advances and open research challenges. *IEEE Access*, 7, 155472–155488.

Ahmed, B. S., Bures, M., Frajtak, K. & Cerny, T. (2019b). Aspects of quality in Internet of Things (IoT) solutions : A systematic mapping study. *IEEE Access*, 7, 13758–13780.

Akram, S. V., Singh, R., AlZain, M. A., Gehlot, A., Rashid, M., Faragallah, O. S., El-Shafai, W. & Prashar, D. (2021). Performance analysis of iot and long-range radio-based sensor node and gateway architecture for solid waste management. *Sensors*, 21(8), 2774.

Al-Debagy, O. & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 000149–000154.

Al-Masri, E. (2018). Enhancing the microservices architecture for the internet of things. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5119–5125.

Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F. & Chaudhry, S. R. (2016). IoT architecture challenges and issues : Lack of standardization. *2016 Future technologies conference (FTC)*, pp. 731–738.

Alenezi, M. & Almustafa, K. (2015). Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology*, 8(2), 257–266.

Alfonso, I., Garcés, K., Castro, H. & Cabot, J. (2021). Self-adaptive architectures in IoT systems : a systematic literature review. *Journal of Internet Services and Applications*, 12(1), 1–28.

Ali, Z. H., Ali, H. A. & Badawy, M. M. (2015). Internet of Things (IoT) : definitions, challenges and recent research directions. *International Journal of Computer Applications*, 128(1), 37–47.

AlOmar, E. A., AlRubaye, H., Mkaouer, M. W., Ouni, A. & Kessentini, M. (2021). Refactoring practices in the context of modern code review : An industrial case study at Xerox. *2021 IEEE/ACM 43rd International Conference on Software Engineering : Software Engineering in Practice (ICSE-SEIP)*, pp. 348–357.

Alreshidi, A. & Ahmad, A. (2019). Architecting software for the internet of thing based systems. *Future Internet*, 11(7), 153.

Alrubaye, H., Alshoaibi, D., Alomar, E., Mkaouer, M. W. & Ouni, A. (2020). How does library migration impact software quality and comprehension ? an empirical study. *International Conference on Software and Software Reuse*, pp. 245–260.

Alshohoumi, F., Sarrab, M., AlHamadani, A. & Al-Abri, D. (2019). Systematic review of existing IoT architectures security and privacy issues and concerns. *International Journal of Advanced Computer Science and Applications*, 10(7), 1–6.

Alshuqayran, N., Ali, N. & Evans, R. (2016). A systematic mapping study in microservice architecture. *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51.

Amazon. (2023). IoT. Repéré à https://aws.amazon.com/fr/what-is/iot/#:~:text=The%20term%20IoT%2C%20or%20Internet,as%20between%20the%20devices%20themselves.

Anda, B. (2007). Assessing software system maintainability using structural measures and expert assessments. *2007 IEEE International Conference on Software Maintenance*, pp. 204–213.

Appleton, B. (1997). Patterns and software : Essential concepts and terminology. *Object Magazine Online*, 3(5), 20–25.

Aquino, G., Queiroz, R., Merrett, G. & Al-Hashimi, B. (2019). The circuit breaker pattern targeted to future iot applications. *International Conference on Service-Oriented Computing*, pp. 390–396.

Arcos-Medina, G. & Mauricio, D. (2019). Aspects of software quality applied to the process of agile software development : a systematic literature review. *International Journal of System Assurance Engineering and Management*, 10, 867–897.

Ashouri, M., Davidsson, P. & Spalazzese, R. (2021). Quality attributes in edge computing for the Internet of Things : A systematic mapping study. *Internet of Things*, 13, 100346.

Atzori, L., Iera, A. & Morabito, G. (2010). The internet of things : A survey. *Computer networks*, 54(15), 2787–2805.

Aziz, M. W., Musharaf, U. & Sayyed, A. (2021). Towards a Software Architecture for Internet of Things based System of Systems. *International Journal*, 9(3), 1-9.

Babun, L., Denney, K., Celik, Z. B., McDaniel, P. & Uluagac, A. S. (2021). A survey on IoT platforms : Communication, security, and privacy perspectives. *Computer Networks*, 192, 108040.

Baccelli, E., Hahm, O., Günes, M., Wählisch, M. & Schmidt, T. C. (2013). RIOT OS : Towards an OS for the Internet of Things. *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 79–80.

Baldini, G., Skarmeta, A., Fourneret, E., Neisse, R., Legeard, B. & Le Gall, F. (2016). Security certification and labelling in Internet of Things. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 627–632.

Baldwin, C. Y. & Clark, K. B. (2000). *Design rules : The power of modularity*. MIT press.

Banijamali, A., Pakanen, O.-P., Kuvaja, P. & Oivo, M. (2020). Software architectures of the convergence of cloud computing and the Internet of Things : A systematic literature review. *Information and Software Technology*, 122, 106271.

Banker, R. D., Davis, G. B. & Slaughter, S. A. (1998). Software development practices, software complexity, and software maintenance performance : A field study. *Management science*, 44(4), 433–450.

Barroca Filho, I. d. M. & de Aquino Junior, G. S. (2018). A software reference architecture for IoT-based healthcare applications. *Computational Science and Its Applications–ICCSA 2018 : 18th International Conference, Melbourne, VIC, Australia, July 2–5, 2018, Proceedings, Part IV 18*, pp. 173–188.

Beck, F. & Diehl, S. (2011). On the congruence of modularity and code coupling. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 354–364.

Belson, B., Holdsworth, J., Xiang, W. & Philippa, B. (2019). A survey of asynchronous programming using coroutines in the Internet of Things and embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(3), 1–21.

Benayache, A., Bilami, A., Barkat, S., Lorenz, P. & Taleb, H. (2019). MsM : A microservice middleware for smart WSN-based IoT application. *Journal of Network and Computer Applications*, 144, 138–154.

Benhamaid, S., Bouabdallah, A. & Lakhlef, H. (2022). Recent advances in energy management for Green-IoT : An up-to-date and comprehensive survey. *Journal of Network and Computer Applications*, 198, 103257.

Berger, E. D., Hollenbeck, C., Maj, P., Vitek, O. & Vitek, J. (2019). On the impact of programming languages on code quality : A reproduction study. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 41(4), 1–24.

Beszédes, Á., Ferenc, R., Gyimóthy, T., Dolenc, A. & Karsisto, K. (2003). Survey of code-size reduction methods. *ACM Computing Surveys (CSUR)*, 35(3), 223–267.

Bojkić, M., Pržulj, d., Stefanović, M. & Ristic, S. (2020). Usage of dependency injection within different frameworks. *19th Int. Symp. INFOTEH-JAHORINAAt-Jahorina*, pp. 18–20.

Borges, H. & Valente, M. T. (2018). What's in a github star ? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146, 112–129.

Bowles, J. B. (2004). Code from requirements : new productivity tools improve the reliability and maintainability of software systems. *Annual Symposium Reliability and Maintainability, 2004-RAMS*, pp. 68–72.

Boyanov, L., Kisimov, V. & Christov, Y. (2020). Evaluating iot reference architecture. *2020 International Conference Automatics and Informatics (ICAI)*, pp. 1–5.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M. & Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4), 571–583.

Bures, M., Klima, M., Rechtberger, V., Ahmed, B. S., Hindy, H. & Bellekens, X. (2021). Review of specific features and challenges in the current internet of things systems impacting their security and reliability. *Trends and Applications in Information Systems and Technologies : Volume 3 9*, pp. 546–556.

Burhan, M., Rehman, R. A., Khan, B. & Kim, B.-S. (2018). IoT elements, layered architectures and security issues : A comprehensive survey. *Sensors*, 18(9), 2796.

Buse, R. P. & Weimer, W. R. (2008). A metric for software readability. *Proceedings of the 2008 international symposium on Software testing and analysis*, pp. 121–130.

Buse, R. P. & Weimer, W. R. (2009). Learning a metric for code readability. *IEEE Transactions on software engineering*, 36(4), 546–558.

Butzin, B., Golatowski, F. & Timmermann, D. (2016). Microservices approach for the internet of things. *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–6.

Campeanu, G. (2018). A mapping study on microservice architectures of Internet of Things and cloud computing solutions. *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4.

Castor Filho, F., Garcia, A. & Rubira, C. M. F. (2007). Extracting error handling to aspects : A cookbook. *2007 IEEE International Conference on Software Maintenance*, pp. 134–143.

Chen, H., Persico, V. & Pescapè, A. (2019). eMES : Easing maintenance of entity services in service oriented software-defined Internet of Things. *2019 Sixth International Conference on Software Defined Systems (SDS)*, pp. 80–87.

Chen, S., Xu, H., Liu, D., Hu, B. & Wang, H. (2014). A vision of IoT : Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things journal*, 1(4), 349–359.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6), 476–493.

Choudhry, A. & Premchand, A. (2021). Microservices and devops for optimal benefits from iot in manufacturing. *Proceedings of International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications*, pp. 375–384.

Clement, S. J., McKee, D. W. & Xu, J. (2017). Service-oriented reference architecture for smart cities. *2017 IEEE symposium on service-oriented system engineering (SOSE)*, pp. 81–85.

Cooke, A., Smith, D. & Booth, A. (2012). Beyond PICO : the SPIDER Tool For Qualitative Evidence Synthesis. *Qualitative health research*, 22(10), 1435–1443.

Corno, F., De Russis, L. & Sáenz, J. P. (2020). How is open source software development different in popular IoT projects ? *IEEE Access*, 8, 28337–28348.

Cppcheck. (2021). Cppcheck. Repéré à https://cppcheck.sourceforge.io/.

Cretu, L.-G. (2012). Smart cities design using event-driven paradigm and semantic web. *Informatica Economica*, 16(4), 57.

Cruz, P., Astudillo, H., Hilliard, R. & Collado, M. (2019). Assessing migration of a 20-year-old system to a micro-service platform using ATAM. *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 174–181.

da Silva, B. C., Maciel, R. S. P. & Ramalho, F. (2013). Evaluating maintainability of MDA software process models. *Product-Focused Software Process Improvement : 14th International Conference, PROFES 2013, Paphos, Cyprus, June 12-14, 2013. Proceedings 14*, pp. 199–213.

Dai, G. & Wang, Y. (2012). Design on architecture of internet of things. Dans *Advances in Computer Science and Information Engineering* (pp. 1–7). Springer.

Datta, S. K. & Bonnet, C. (2018). Next-generation, data centric and end-to-end iot architecture based on microservices. *2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 206–212.

De Iasio, A., Furno, A., Goglia, L. & Zimeo, E. (2019). A microservices platform for monitoring and analysis of iot traffic data in smart cities. *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5223–5232.

de Santana, C. J. L., de Mello Alencar, B. & Prazeres, C. V. S. (2019). Reactive microservices for the internet of things : A case study in fog computing. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 1243–1251.

Detterer, P., Nabi, M., Jiao, H. & Basten, T. (2022). Receiver Design With an Adjustable Energy-Signal-Quality Tradeoff for IoT Networks. *IEEE Internet of Things Journal*, 9(22), 23086–23096.

Díaz López, D., Blanco Uribe, M., Santiago Cely, C., Tarquino Murgueitio, D., Garcia Garcia, E., Nespoli, P. & Gómez Mármol, F. (2018). Developing secure IoT services : A security-oriented review of IoT platforms. *Symmetry*, 10(12), 669.

Dineva, K. & Atanasova, T. (2020). Architectural ML framework for IoT services delivery based on microservices. *International Conference on Distributed Computer and Communication Networks*, pp. 698–711.

Dobaj, J., Iber, J., Krisper, M. & Kreiner, C. (2018). A microservice architecture for the industrial internet-of-things. *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, pp. 1–15.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L. (2017). Microservices : yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195–216.

Driss, M., Hasan, D., Boulila, W. & Ahmad, J. (2021). Microservices in IoT security : current solutions, research challenges, and future directions. *Procedia Computer Science*, 192, 2385–2395.

Du Bois, B., Demeyer, S. & Verelst, J. (2004). Refactoring-improving coupling and cohesion of existing code. *11th working conference on reverse engineering*, pp. 144–151.

du Plessis, S., Mendes, B. & Correia, N. (2021). A Comparative Study of Microservices Frameworks in IoT Deployments. *2021 International Young Engineers Forum (YEF-ECE)*, pp. 86–91.

Dunford, R., Su, Q. & Tamang, E. (2014). The pareto principle. 1–9.

Dyba, T., Dingsoyr, T. & Hanssen, G. K. (2007). Applying systematic reviews to diverse study types : An experience report. *First international symposium on empirical software engineering and measurement (ESEM 2007)*, pp. 225–234.

e Abreu, F. B. & Goulao, M. (2001). Coupling and cohesion as modularization drivers : Are we being over-persuaded ? *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pp. 47–57.

Ebert, C., Cain, J., Antoniol, G., Counsell, S. & Laplante, P. (2016). Cyclomatic complexity. *IEEE software*, 33(6), 27–29.

eclipse ditto. (2021). eclipse-ditto/ditto. Repéré à https://github.com/eclipse-ditto/ditto.

Edler von Koch, T. J., Franke, B., Bhandarkar, P. & Dasgupta, A. (2014). Exploiting function similarity for code size reduction. *Proceedings of the 2014 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, pp. 85–94.

El Khalyly, B., Belangour, A., Banane, M. & Erraissi, A. (2020a). A comparative study of microservices-based iot platforms. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(7), 389–398.

El Khalyly, B., Belangour, A., Erraissi, A. & Banane, M. (2020b). Devops and microservices based internet of things meta-model. *International Journal*, 8(9), 1-9.

Fahmideh, M. & Zowghi, D. (2018). IoT smart city architectures : An analytical evaluation. *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 709–715.

Ferry, N., Solberg, A., Song, H., Lavirotte, S., Tigli, J.-Y., Winter, T., Muntés-Mulero, V., Metzger, A., Rios Velasco, E. & Castelruiz Aguirre, A. (2019). Enact : Development, operation, and quality assurance of trustworthy smart iot systems. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment : First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1*, pp. 112–127.

Fowler, M. (2018). *Refactoring*. Addison-Wesley Professional.

Gama, K., Touseau, L. & Donsez, D. (2012). Combining heterogeneous service technologies for building an Internet of Things middleware. *Computer Communications*, 35(4), 405–417.

Garlan, D. & Perry, D. E. (1995). Introduction to the special issue on software architecture. *IEEE Trans. Software Eng.*, 21(4), 269–274.

Gaur, A. S., Budakoti, J. & Lung, C.-H. (2018). Design and performance evaluation of containerized microservices on edge gateway in mobile iot. *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 138–145.

Gavrilović, N. & Mishra, A. (2021). Software architecture of the internet of things (IoT) for smart city, healthcare and agriculture : analysis and improvement directions. *Journal of Ambient Intelligence and Humanized Computing*, 12(1), 1315–1336.

Gergel, B., Stroulia, E., Smit, M. & Hoover, H. J. (2011). Maintainability and Source Code Conventions : An Analysis of Open Source Projects. 1-9.

Gill, A. Q., Behbood, V., Ramadan-Jradi, R. & Beydoun, G. (2017). Iot architectural concerns : a systematic review. *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, pp. 1-9.

Giray, G., Tekinerdogan, B. & Tüzün, E. (2017). IoT system development methods. *Internet of Things : Challenges, Advances and Applications*, 141–159.

Gorbachenko, I., Gorshkov, E. & Filipkina, T. (2020). Application of various metrics to assess the program code quality. *Journal of Physics : Conference Series*, 1679(3), 032087.

Greengard, S. (2021). *The internet of things*. MIT press.

Grønbæk, I. (2008). Architecture for the Internet of Things (IoT) : API and interconnect. *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*, pp. 802–807.

Gubbi, J., Buyya, R., Marusic, S. & Palaniswami, M. (2013). Internet of Things (IoT) : A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7), 1645–1660.

Guerrero-Ulloa, G., Rodríguez-Domínguez, C. & Hornos, M. J. (2023). Agile methodologies applied to the development of Internet of Things (IoT)-based systems : A review. *Sensors*, 23(2), 790.

Gupta, B. B. & Quamara, M. (2020). An overview of Internet of Things (IoT) : Architectural aspects, challenges, and protocols. *Concurrency and Computation : Practice and Experience*, 32(21), e4946.

Guşeilă, L. G., Bratu, D.-V. & Moraru, S.-A. (2019). Continuous testing in the development of iot applications. *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, pp. 1–6.

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F. & Reinfurt, L. (2016). Comparison of IoT platform architectures : A field study based on a reference architecture. *2016 Cloudification of the Internet of Things (CIoT)*, pp. 1–6.

Hanson, D. R. (1983). Simple code optimizations. *Software : Practice and Experience*, 13(8), 745–763.

Hanusz, Z., Tarasinska, J. & Zielinski, W. (2016). Shapiro–Wilk test with known mean. *REVSTAT-Statistical Journal*, 14(1), 89–100.

Hawkins, D. M. (1980). *Identification of outliers*. Springer.

Hegedus, P. (2013). Revealing the effect of coding practices on software maintainability. *2013 ieee international conference on software maintenance*, pp. 578–581.

Heitlager, I., Kuipers, T. & Visser, J. (2007). A practical model for measuring maintainability. *6th international conference on the quality of information and communications technology (QUATIC 2007)*, pp. 30–39.

Hu, Y.-L., Cho, Y.-Y., Su, W.-B., Wei, D. S., Huang, Y., Chen, J.-L., Chen, I.-Y. & Kuo, S.-Y. (2015). A programming framework for implementing fault-tolerant mechanism in iot applications. *Algorithms and Architectures for Parallel Processing : 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*, pp. 771–784.

Ibrahim, H., Mostafa, N., Halawa, H., Elsalamouny, M., Daoud, R., Amer, H., Adel, Y., Shaarawi, A., Khattab, A. & ElSayed, H. (2019). A layered IoT architecture for greenhouse monitoring and remote control. *SN Applied Sciences*, 1, 1–12.

Ihirwe, F., Di Ruscio, D., Gianfranceschi, S. & Pierantonio, A. (2022). Assessing the Quality of Low-Code and Model-driven Engineering Platforms for Engineering IoT Systems. *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pp. 583–594.

Innerbichler, J., Gonul, S., Damjanovic-Behrendt, V., Mandler, B. & Strohmeier, F. (2017). NIMBLE collaborative platform : Microservice architectural approach to federated IoT. *2017 Global Internet of Things Summit (GIoTS)*, pp. 1–6.

Ismail, A. A., Hamza, H. S. & Kotb, A. M. (2018). Performance evaluation of open source IoT platforms. *2018 IEEE global conference on internet of things (GCIoT)*, pp. 1–5.

ISO. (2003). ISO/IEC 9126 [Format]. Repéré à https://www.iso.org/fr/standard/22891.html.

ISO. (2004). ISO25000 [Format]. Repéré à https://www.iso.org/fr/standard/64764.html.

Jacob, P. M. & Mani, P. (2018). Software architecture pattern selection model for Internet of Things based systems. *IET Software*, 12(5), 390–396.

Jakeman, A. J., Letcher, R. A. & Norton, J. P. (2006). Ten iterative steps in development and evaluation of environmental models. *Environmental Modelling & Software*, 21(5), 602–614.

Jarwar, M. A., Ali, S., Kibria, M. G., Kumar, S. & Chong, I. (2017). Exploiting interoperable microservices in web objects enabled Internet of Things. *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 49–54.

Jarwar, M. A., Kibria, M. G., Ali, S. & Chong, I. (2018). Microservices in web objects enabled IoT environment for enhancing reusability. *Sensors*, 18(2), 352.

Javadpour, A., Wang, G. & Rezaei, S. (2020). Resource management in a peer to peer cloud network for IoT. *Wireless Personal Communications*, 115, 2471–2488.

Jia, M., Komeily, A., Wang, Y. & Srinivasan, R. S. (2019). Adopting Internet of Things for the development of smart buildings : A review of enabling technologies and applications. *Automation in Construction*, 101, 111–126.

Jin, Z., Xing, L., Fang, Y., Jia, Y., Yuan, B. & Liu, Q. (2022). P-Verifier : Understanding and Mitigating Security Risks in Cloud-based IoT Access Policies. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1647–1661.

Jovanovic, B. (2023). Internet of Things statistics for 2023 – Taking Things Apart. Repéré à https://dataprot.net/statistics/iot-statistics/.

Jun, H. K. & Rana, M. E. (2021). Evaluating the Impact of Design Patterns on Software Maintainability : An Empirical Evaluation. *2021 Third International Sustainability and Resilience Conference : Climate Change*, pp. 539–548.

Jung, E., Cho, I. & Kang, S. M. (2014). An agent modeling for overcoming the heterogeneity in the IoT with design patterns. *Mobile, Ubiquitous, and Intelligent Computing : MUSIC 2013*, pp. 69–74.

Kamaludeen, N. B. A., Lee, S. P. & Parizi, R. M. (2019). Guideline-based approach for IoT home application development. *2019 international conference on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, pp. 929–936.

Kang, R., Zhou, Z., Liu, J., Zhou, Z. & Xu, S. (2018). Distributed monitoring system for microservices-based iot middleware system. *International Conference on Cloud Computing and Security*, pp. 467–477.

Kashyap, N., Rana, A., Kansal, V. & Walia, H. (2021). Improve cloud based iot architecture layer security-a literature review. *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 772–777.

Kazanavičius, J. & Mažeika, D. (2019). Migrating legacy software to microservices architecture. *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pp. 1–5.

Kearney, J. P., Sedlmeyer, R. L., Thompson, W. B., Gray, M. A. & Adler, M. A. (1986). Software complexity measurement. *Communications of the ACM*, 29(11), 1044–1050.

Khan, R., Khan, S. U., Zaheer, R. & Khan, S. (2012). Future internet : the internet of things architecture, possible applications and key challenges. *2012 10th international conference on frontiers of information technology*, pp. 257–260.

Khanda, K., Salikhov, D., Gusmanov, K., Mazzara, M. & Mavridis, N. (2017). Microservice-based iot for smart buildings. *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 302–308.

Kim, M., Park, J. H. & Lee, N. Y. (2017). A quality model for IoT service. *Advances in Computer Science and Ubiquitous Computing : CSA-CUTE2016 8*, pp. 497–504.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004), 1–26.

Kitchenham, B. & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. 1-9.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J. & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1), 7–15.

Klima, M., Rechtberger, V., Bures, M., Bellekens, X., Hindy, H. & Ahmed, B. S. (2021). Quality and reliability metrics for IoT systems : a consolidated view. *Science and Technologies for Smart Cities : 6th EAI International Conference, SmartCity360°, Virtual Event, December 2-4, 2020, Proceedings*, pp. 635–650.

Klima, M., Bures, M., Frajtak, K., Rechtberger, V., Trnka, M., Bellekens, X., Cerny, T. & Ahmed, B. S. (2022). Selected Code-Quality Characteristics and Metrics for Internet of Things Systems. *IEEE Access*, 10, 46144–46161.

Kmet, L. M., Cook, L. S. & Lee, R. C. (2004). Standard quality assessment criteria for evaluating primary research papers from a variety of fields. 1-9.

Kohar, R. (2020). IoT systems based on SOA services : Methodologies, Challenges and Future directions. *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 556–560.

Koroniotis, N., Moustafa, N., Schiliro, F., Gauravaram, P. & Janicke, H. (2021). The SAir-IIoT Cyber Testbed as a Service : A Novel Cybertwins Architecture in IIoT-Based Smart Airports. *IEEE Transactions on Intelligent Transportation Systems*, 1–9.

Krivic, P., Skocir, P., Kusek, M. & Jezic, G. (2017). Microservices as agents in IoT systems. *KES International Symposium on Agent and Multi-Agent Systems : Technologies and Applications*, pp. 22–31.

Krylovskiy, A., Jahn, M. & Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. *2015 3rd international conference on future internet of things and cloud*, pp. 25–30.

Kugler, S., Czwick, C., Nguyen, H. P. L. & Anderl, R. (2021). Method for the Targeted Selection and Installation of an IoT-Platform. *International Conference on Applied Human Factors and Ergonomics*, pp. 116–123.

Kuila, S., Dhanda, N., Joardar, S. & Neogy, S. (2019). Analytical survey on standards of Internet of Things framework and platforms. *Emerging Technologies in Data Mining and Information Security : Proceedings of IEMIS 2018, Volume 3*, pp. 33–44.

Kumar, K., Mouli, C. & Kumar, U. (2017). A survey on the Internet of Things-based service orientated architecture. *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pp. 435–439.

kymjs. (2021). kymjs/CJFrameForAndroid. Repéré à https://github.com/kymjs/CJFrameForAndroid/blob/master/cjframe/src/org/kymjs/cjframe/bean/AndroidPackage.java.

Kyriazopoulou, C. (2015). Smart city technologies and architectures : A literature review. *2015 International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, pp. 1–12.

Lai, C., Boi, F., Buschettu, A. & Caboni, R. (2019). IoT and microservice architecture for multimobility in a smart city. *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 238–242.

Lajios, G., Schmedding, D. & Volmering, F. (2008). Supporting language conversion by metric based reports. *2008 12th European Conference on Software Maintenance and Reengineering*, pp. 314–316.

Larrucea, X., Combelles, A., Favaro, J. & Taneja, K. (2017). Software engineering for the internet of things. *IEEE Software*, 34(1), 24–28.

LCOM. (2021). LCOM. Repéré à http://www.virtualmachinery.com/jhawkmetricsclass.htm.

Lefurgy, C., Piccininni, E. & Mudge, T. (2000). Reducing code size with run-time decompression. *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550)*, pp. 218–228.

Li, B. (2019). Efficiency optimization for communication service based on qos technology. *IEEE Access*, 7, 48838–48848.

Li, W. (1998). Another metric suite for object-oriented programming. *Journal of Systems and Software*, 44(2), 155–162.

Liao, Z., Nazir, S., Khan, H. U. & Shafiq, M. (2021). Assessing security of software components for Internet of Things : a systematic review and future directions. *Security and Communication Networks*, 2021, 1–22.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H. & Zhao, W. (2017). A survey on internet of things : Architecture, enabling technologies, security and privacy, and applications. *IEEE internet of things journal*, 4(5), 1125–1142.

Lu, D., Huang, D., Walenstein, A. & Medhi, D. (2017). A secure microservice framework for iot. *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 9–18.

MacFarland, T. W., Yates, J. M., MacFarland, T. W. & Yates, J. M. (2016). Mann–whitney u test. *Introduction to nonparametric statistics for the biological sciences using R*, 103–132.

Madakam, S., Lake, V., Lake, V., Lake, V. et al. (2015). Internet of Things (IoT) : A literature review. *Journal of Computer and Communications*, 3(05), 164.

Magaia, N., Gomes, P., Silva, L., Sousa, B., Mavromoustakis, C. X. & Mastorakis, G. (2020). Development of mobile IoT solutions : approaches, architectures, and methodologies. *IEEE Internet of Things Journal*, 8(22), 16452–16472.

Mahmoud, R., Yousuf, T., Aloul, F. & Zualkernan, I. (2015). Internet of things (IoT) security : Current status, challenges and prospective measures. *2015 10th international conference for internet technology and secured transactions (ICITST)*, pp. 336–341.

Márquez, G., Astudillo, H. & Kazman, R. (2023). Architectural tactics in software architecture : A systematic mapping study. *Journal of Systems and Software*, 197, 111558.

Martins, L. M. e., Filho, F. L. d. C., Júnior, R. T. d. S., Giozza, W. F. & da Costa, J. P. C. (2017). Increasing the dependability of iot middleware with cloud computing and microservices. *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 203–208.

Mateo-Fornés, J., Pagès-Bernaus, A., Plà-Aragonés, L. M., Castells-Gasia, J. P. & Babot-Gaspa, D. (2021). An internet of things platform based on microservices and cloud paradigms for livestock. *Sensors*, 21(17), 5949.

Maurya, L. (2010). Comparison of software architecture evaluation methods for software quality attributes. *Journal of Global Research in Computer Science*, 1(4), 1–5.

Mayer, C. (2022). *The Art of Clean Code : Best Practices to Eliminate Complexity and Simplify Your Life*. No Starch Press.

Meloni, A., Pegoraro, P. A., Atzori, L., Benigni, A. & Sulis, S. (2018). Cloud-based IoT solution for state estimation in smart grids : Exploiting virtualization and edge-intelligence technologies. *Computer networks*, 130, 156–165.

Minani, J. B., Sabir, F., Moha, N. & Guéhéneuc, Y.-G. (2024). A Multimethod Study of Internet of Things Systems Testing in Industry. *IEEE Internet of Things Journal*, 11(1), 1662-1684. doi : 10.1109/JIOT.2023.3291233.

Miorandi, D., Sicari, S., De Pellegrini, F. & Chlamtac, I. (2012). Internet of things : Vision, applications and research challenges. *Ad hoc networks*, 10(7), 1497–1516.

Mishra, K. N. & Chakraborty, C. (2020). A novel approach toward enhancing the quality of life in smart cities using clouds and IoT-based technologies. *Digital Twin Technologies and Smart Cities*, 19–35.

Moadad, N., Damaj, I. & El Kabani, I. (2022). A Generic MDA-IoT Architecture for Connected Vehicles in Smart Cities. *2022 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, pp. 122–129.

Mrabet, H., Belguith, S., Alhomoud, A. & Jemai, A. (2020). A survey of IoT security based on a layered architecture of sensing and data analysis. *Sensors*, 20(13), 3625.

Muccini, H. & Moghaddam, M. T. (2018a). Iot architectural styles. *European Conference on Software Architecture*, pp. 68–85.

Muccini, H. & Moghaddam, M. T. (2018b). Iot architectural styles. *European Conference on Software Architecture*, pp. 68–85.

Muccini, H., Spalazzese, R., Moghaddam, M. T. & Sharaf, M. (2018). Self-adaptive IoT architectures : An emergency handling case study. *Proceedings of the 12th European Conference on Software Architecture : Companion Proceedings*, pp. 1–6.

Mullaney, C., Aijaz, A., Sealey, N. & Holden, B. (2022). Peer-to-Peer Energy Trading meets IOTA : Toward a Scalable, Low-Cost, and Efficient Trading System. *arXiv preprint arXiv :2210.06427*, 1-9.

Multimetric. (2021). Multimetric. Repéré à https://github.com/priv-kweihmann/multimetric.

Muratkar, T. S., Bhurane, A. & Kothari, A. (2020). Battery-less internet of things–A survey. *Computer Networks*, 180, 107385.

Nazar, N., Hu, Y. & Jiang, H. (2016). Summarizing software artifacts : A literature review. *Journal of Computer Science and Technology*, 31(5), 883–909.

Neelamegam, C. & Punithavalli, M. (2009). A survey-object oriented quality metrics. *Global Journal of Computer Science and Technology*, 9(4), 183–186.

Nicholson, A. J., Chawathe, Y., Chen, M. Y., Noble, B. D. & Wetherall, D. (2006). Improved access point selection. *Proceedings of the 4th international conference on Mobile systems, applications and services*, pp. 233–245.

Ortiz, G., Boubeta-Puig, J., Criado, J., Corral-Plaza, D., Garcia-de Prado, A., Medina-Bulo, I. & Iribarne, L. (2022). A microservice architecture for real-time IoT data processing : A reusable Web of things approach for smart ports. *Computer Standards & Interfaces*, 81, 103604.

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E. et al. (2021). The PRISMA 2020 Statement : An Updated Guideline For Reporting Systematic Reviews. *International journal of surgery*, 88, 105906.

Pallewatta, S., Kostakos, V. & Buyya, R. (2019). Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments. *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 71–81.

Parnas, D. L. & Würges, H. (1976). Response to undesired events in software systems. *Proceedings of the 2nd international conference on Software engineering*, pp. 437–446.

Paul, B. (2022). Internet of Things (IoT), Three-Layer Architecture, Security Issues and Counter Measures. Dans *ICT Analysis and Applications* (pp. 23–34). Springer.

Pena, P. A., Sarkar, D. & Maheshwari, P. (2015). A big-data centric framework for smart systems in the world of internet of everything. *2015 international conference on computational science and computational intelligence (CSCI)*, pp. 306–311.

Piantadosi, V., Fierro, F., Scalabrino, S., Serebrenik, A. & Oliveto, R. (2020). How does code readability change during software evolution ? *Empirical Software Engineering*, 25, 5374–5412.

Politowski, C., Petrillo, F., Montandon, J. E., Valente, M. T. & Guéhéneuc, Y.-G. (2021). Are game engines software frameworks ? a three-perspective study. *Journal of Systems and Software*, 171, 110846.

Porruvecchio, G., Romanino, A., Casari, C. & Sanna, R. (2021). A microservice-based platform for IoT application development. *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0332–0336.

Posnett, D., Hindle, A. & Devanbu, P. (2011). A simpler model of software readability. *Proceedings of the 8th working conference on mining software repositories*, pp. 73–82.

Power, A. & Kotonya, G. (2018). A microservices architecture for reactive and proactive fault tolerance in iot systems. *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 588–599.

Prasandy, T., Murad, D. F., Darwis, T. et al. (2020). Migrating application from monolith to microservices. *2020 International Conference on Information Management and Technology (ICIMTech)*, pp. 726–731.

Qin, Z., Zheng, X. & Xing, J. (2008). *Introduction to software architecture*. Springer.

Radwan, M. & Heckel, R. (2015). Detecting and Refactoring Operational Smells within the Domain Name System. *arXiv preprint arXiv :1504.02615*, 1–9.

Randell, B. & Xu, J. (1995). The evolution of the recovery block concept. *Software fault tolerance*, 3, 1–22.

Rao, A., Schelén, O. & Lindgren, A. (2016). Performance implications for IoT over information centric networks. *Proceedings of the eleventh ACM workshop on challenged networks*, pp. 57–62.

Ray, P. P. (2018). A survey on Internet of Things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3), 291–319.

Razzaq, A. (2020). A systematic review on software architectures for IoT systems and future direction to the adoption of microservices architecture. *SN Computer Science*, 1(6), 1–30.

Saadeh, H., Almobaideen, W. & Sabri, K. E. (2017). Internet of Things : A review to support IoT architecture's design. *2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)*, pp. 1–7.

Said, O. & Tolba, A. (2016). Performance evaluation of a dual coverage system for internet of things environments. *Mobile Information Systems*, 2016, 1–9.

Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M. & Al-Hammadi, Y. (2018). IoT applications : From mobile agents to microservices architecture. *2018 International conference on innovations in information technology (IIT)*, pp. 117–122.

Saleem, M. U., Usman, M. R., Usman, M. A. & Politis, C. (2022). Design, deployment and performance evaluation of an IoT based smart energy management system for demand side management in smart grid. *IEEE Access*, 10, 15261–15278.

Samoladas, I., Stamelos, I., Angelis, L. & Oikonomou, A. (2004). Open source software development should strive for even greater code maintainability. *Communications of the ACM*, 47(10), 83–87.

Sampaio, I. B. & Barbosa, L. (2016). Software readability practices and the importance of their teaching. *2016 7th International Conference on Information and Communication Systems (ICICS)*, pp. 304–309.

Santana, C., Alencar, B. & Prazeres, C. (2018). Microservices : A mapping study for internet of things solutions. *2018 IEEE 17th international symposium on network computing and applications (NCA)*, pp. 1–4.

Santana, C., Andrade, L., Mello, B., Batista, E., Sampaio, J. V. & Prazeres, C. (2019). A reliable architecture based on reactive microservices for IoT applications. *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*, pp. 15–19.

Santana, C., Andrade, L., Delicato, F. C. & Prazeres, C. (2021). Increasing the availability of IoT applications with reactive microservices. *Service Oriented Computing and Applications*, 15(2), 109–126.

Santos, L., Silva, E., Batista, T., Cavalcante, E., Leite, J. & Oquendo, F. (2020). An architectural style for internet of things systems. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 1488–1497.

Saovapakhiran, B., Naruephiphat, W., Charnsripinyo, C., Baydere, S. & Ozdemir, S. (2022). QoE-Driven IoT Architecture : A Comprehensive Review on System and Resource Management. *IEEE Access*, 1–9.

Sarkar, C., SN, A. U. N., Prasad, R. V., Rahim, A., Neisse, R. & Baldini, G. (2014). DIAT : A scalable distributed architecture for IoT. *IEEE Internet of Things journal*, 2(3), 230–239.

Scalabrino, S., Linares-Vasquez, M., Poshyvanyk, D. & Oliveto, R. (2016). Improving code readability models with textual features. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pp. 1–10.

Scalabrino, S., Linares-Vásquez, M., Oliveto, R. & Poshyvanyk, D. (2018). A comprehensive model for code readability. *Journal of Software : Evolution and Process*, 30(6), e1958.

Schneberger, S. L. & McLean, E. R. (2003). The complexity cross : implications for practice. *Communications of the ACM*, 46(9), 216–225.

SciTools. (2021). Understand. Repéré à https://www.scitools.com/.

Sedano, T. (2016). Code readability testing, an empirical study. *2016 IEEE 29th International conference on software engineering education and training (CSEET)*, pp. 111–117.

Sethi, P. & Sarangi, S. R. (2017). Internet of things : architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017, 1-9.

Sharma, A., Kumar, M. & Agarwal, S. (2015). A complete survey on software architectural styles and patterns. *Procedia Computer Science*, 70, 16–28.

Sicari, S., Cappiello, C., De Pellegrini, F., Miorandi, D. & Coen-Porisini, A. (2016). A security-and quality-aware system architecture for Internet of Things. *Information Systems Frontiers*, 18, 665–677.

Siddiqui, H., Khendek, F. & Toeroe, M. (2023). Microservices based architectures for IoT systems-State-of-the-art review. *IEEE Internet of Things*, 100854.

Silva, A. F. d., de Lima, B. N. & Pereira, F. M. Q. (2021). Exploring the space of optimization sequences for code-size reduction : Insights and tools. *Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction*, pp. 47–58.

Singh, M. & Baranwal, G. (2018). Quality of service (qos) in internet of things. *2018 3rd International Conference On Internet of Things : Smart Innovation and Usages (IoT-SIU)*, pp. 1–6.

Sithole, V. & Marshall, L. (2020). Systematic methods for organising patterns for the internet of things : A preliminary exploration. *Internet of Things*, 11, 100268.

Soetens, Q. D. & Demeyer, S. (2010). Studying the effect of refactorings : a complexity metrics perspective. *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pp. 313–318.

Soumyalatha, S. G. H. (2016). Study of IoT : understanding IoT architecture, applications, issues and challenges. *1st International Conference on Innovations in Computing & Net-working (ICICN16), CSE, RRCE. International Journal of Advanced Networking & Applications*, 478, 1-9.

standars. (2021). Statics. Repéré à https://www.mitre.org/sites/default/files/publications/17-1332-best-practices-for-technical-standard-creation.pdf.

Stefanova-Stoyanova, V., Stoyanov, K. & Danov, P. (2021). Comparison Of Architectural Models Of IoT Systems-Advantages And Disadvantages. *2021 XXX International Scientific Conference Electronics (ET)*, pp. 1–5.

Stojanov, Z. & Dobrilovic, D. (2021). Software architecture quality attributes of a layered sensor-based iot system. *IV Bychkov, D. Karastoyanov (Eds.), Proceedings of the 4th Workshop Information Technologies : Algorithms, Models, Systems (ITAMS 2021)*, pp. 66.

Sun, C.-a., Wang, J., Guo, J., Wang, Z. & Duan, L. (2019). A reconfigurable microservice-based migration technique for IoT systems. *International Conference on Service-Oriented Computing*, pp. 142–155.

Sun, X., Liang, Y. & Huang, H. (2020). Design and Implementation of Internet of Things Platform based on Microservice and Lightweight Container. *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 9, 1353–1357.

Sun, Y. & Bai, L. (2021). Examination on Security Performance Analysis Model of Internet of Things Assigned Based on Composite Security Key. *Wireless Communications and Mobile Computing*, 2021, 1–14.

Taherizadeh, S., Stankovski, V. & Grobelnik, M. (2018). A capillary computing architecture for dynamic internet of things : Orchestration of microservices from edge devices to fog and cloud providers. *Sensors*, 18(9), 2938.

Taivalsaari, A. & Mikkonen, T. (2017). A roadmap to the programmable world : software challenges in the IoT era. *IEEE software*, 34(1), 72–80.

Tan, L. & Wang, N. (2010). Future internet : The internet of things. *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, 5, V5–376.

Tanganelli, G., Vallati, C. & Mingozzi, E. (2018). Ensuring quality of service in the internet of things. *New advances in the internet of things*, 139–163.

Tekinerdogan, B. & Köksal, Ö. (2018). Pattern based integration of internet of things systems. *International Conference on Internet of Things*, pp. 19–33.

Tekinerdogan, B., Köksal, Ö. & Çelik, T. (2023). System Architecture Design of IoT-Based Smart Cities. *Applied Sciences*, 13(7), 4173.

Thomas, M. O., Onyimbo, B. A. & Logeswaran, R. (2016). Usability evaluation criteria for internet of things. *Int J Inf Technol Comput Sci*, 8, 10–18.

Tian, H., Xu, X., Lin, T., Cheng, Y., Qian, C., Ren, L. & Bilal, M. (2021). DIMA : Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning. *World Wide Web*, 1–24.

Tran, N. K., Sheng, Q. Z., Babar, M. A., Yao, L., Zhang, W. E. & Dustdar, S. (2019). Internet of Things search engine. *Communications of the ACM*, 62(7), 66–73.

Trilles, S., González-Pérez, A. & Huerta, J. (2020). An IoT platform based on microservices and serverless paradigms for smart farming purposes. *Sensors*, 20(8), 2418.

Tun, S. Y. Y., Madanian, S. & Mirza, F. (2021). Internet of things (IoT) applications for elderly care : a reflective review. *Aging clinical and experimental research*, 33, 855–867.

Udoh, I. S. & Kotonya, G. (2018). Developing IoT applications : challenges and frameworks. *IET Cyber-Physical Systems : Theory & Applications*, 3(2), 65–72.

Uviase, O. & Kotonya, G. (2018). IoT architectural framework : connection and integration framework for IoT systems. *arXiv preprint arXiv :1803.04780*, 1-9.

Valle, P. H. D., Garcés, L. & Nakagawa, E. Y. (2019). A typology of architectural strategies for interoperability. *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse*, pp. 3–12.

Velepucha, V. & Flores, P. (2021). Monoliths to microservices-Migration Problems and Challenges : A SMS. *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, pp. 135–142.

Venkatesh, J., Chan, C., Akyurek, A. S. & Rosing, T. S. (2016). A modular approach to context-aware iot applications. *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 235–240.

Vermeire, T., Faes, J., De Bruyn, P. & Verelst, J. (2019). On the modular structure and evolvability of the Internet of Things architectures. *Patterns 2019 : the eleventh International Conferences on Pervasive Patterns and Applications, May 5-9, 2019, Venice, Italy/Mannaert, Herwig [edit.]*, pp. 17–22.

Viggiato, M., Terra, R., Rocha, H., Valente, M. T. & Figueiredo, E. (2018). Microservices in practice : A survey study. *arXiv preprint arXiv :1808.04836*, 1-9.

Vila, M., Sancho, M.-R. & Teniente, E. (2020). XYZ monitor : IoT monitoring of infrastructures using microservices. *International Conference on Service-Oriented Computing*, pp. 472–484.

Visser, J., Rigal, S., Wijnholds, G., Van Eck, P. & van der Leek, R. (2016). *Building Maintainable Software, C# Edition : Ten Guidelines for Future-Proof Code*. " O'Reilly Media, Inc.".

Vresk, T. & Čavrak, I. (2016). Architecture of an interoperable IoT platform based on microservices. *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1196–1201.

Vural, H., Koyuncu, M. & Guney, S. (2017). A systematic literature review on microservices. *International Conference on Computational Science and Its Applications*, pp. 203–217.

Walls, C. & Breidenbach, R. (2007). *Spring In Action, Updated For Spring 2.0*. Dreamtech Press.

Wang, X., Pollock, L. & Vijay-Shanker, K. (2011). Automatic segmentation of method code into meaningful blocks to improve readability. *2011 18th Working Conference on Reverse Engineering*, pp. 35–44.

Wang, Z., Sun, C.-a. & Aiello, M. (2021). Lightweight and Context-aware Modeling of Microservice-based Internet of Things. *2021 IEEE International Conference on Web Services (ICWS)*, pp. 282–292.

Waris, Z., Jaleel, A., Shoaib, M., Nigar, N. & Abalo, D. (2022). A Suite of Design Quality Metrics for Internet of Things by Modelling Its Ecosystem as a Schema Graph. *Mathematical Problems in Engineering*, 2022, 1-9.

Waseem, M., Liang, P., Márquez, G. & Di Salle, A. (2020). Testing microservices architecture-based applications : A systematic mapping study. *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 119–128.

Washizaki, H., Ogata, S., Hazeyama, A., Okubo, T., Fernandez, E. B. & Yoshioka, N. (2020). Landscape of architecture and design patterns for iot systems. *IEEE Internet of Things Journal*, 7(10), 10091–10101.

Weinberg, G. M. (1971). *The psychology of computer programming*. Van Nostrand Reinhold New York.

Weyrich, M. & Ebert, C. (2015). Reference architectures for the internet of things. *IEEE Software*, 33(1), 112–116.

Wilde, N., Gonen, B., El-Sheikh, E. & Zimmermann, A. (2016). Approaches to the evolution of SOA systems. *Emerging trends in the evolution of service-oriented and enterprise architectures*, 5–21.

Yu, R., Kilari, V. T., Xue, G. & Yang, D. (2019). Load balancing for interdependent IoT microservices. *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 298–306.

Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J. & Yang, X. (2017). A survey on the edge computing for the Internet of Things. *IEEE access*, 6, 6900–6919.

Zeinab, K. A. M. & Elmustafa, S. A. A. (2017). Internet of things applications, challenges and related future technologies. *World Scientific News*, 67(2), 126–148.

Zhang, L., Yuan, H., Chang, S.-H. & Lam, A. (2020a). Research on the overall architecture of Internet of Things middleware for intelligent industrial parks. *The International Journal of Advanced Manufacturing Technology*, 107(3), 1081–1089.

Zhang, X., Wen, Z., Wu, Y. & Zou, J. (2011). The implementation and application of the internet of things platform based on the REST architecture. *2011 International Conference on Business Management and Electronic Information*, 2, 43–45.

Zhang, Y., Sun, S., Zhang, D., Qiu, J. & Tian, Z. (2020b). A consistency-guaranteed approach for Internet of Things software refactoring. *International Journal of Distributed Sensor Networks*, 16(1), 1550147720901680.

Zhen, Y., Zeng, L., Chen, X., Li, X. & Liu, J. (2011). Study of architecture of power Internet of Things. *IET International Conference on Communication Technology and Application (ICCTA 2011)*, pp. 718–722.