Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Аналіз методологій розробки IoT-систем з використанням Agile-технологій.

Виконав(ла): студент(ка) 6 курсу, групи СНмд-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Щеснюк Т.О.
(підпис) (прізвище та ініціали)

Керівник Марценко С.В.
(підпис) (прізвище та ініціали)

Нормоконтроль Дуда О.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
"26" грудня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Щеснюк Тетяна Омелянівна
(прізвище, ім'я, по батькові)

1. Тема роботи **Аналіз методологій розробки IoT-систем з використанням Agile-технологій.**

Керівник роботи **к.т.н., доц. Марценко С.В.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від "*29*" *листопада* 2024 року № *4/7-1142*

2. Термін подання студентом завершеної роботи 26 грудня 2024 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП; 1 ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ;1.1 Огляд стану проблеми;1.2 Сучасні методи розробки IoT-систем;2 ЕТАПИ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ IoT-СИСТЕМ; 2.1 Стандарти, що визначають етапи та процеси життєвого циклу програмного забезпечення; 2.2 Методології розробки IoT на основі Agile Manifesto; 2.3 Моделювання як ключ у методології розробки IoT; 2.4 Методології традиційної розробки, застосовані до розробки IoT; 3 МЕТОДОЛОГІЇ, ІНСТРУМЕНТИ ТА ФРЕЙМВОРКИ, ОРІЄНТОВАНІ НА ПРОЕКТУВАННЯ ТА СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ IoT; 3.1 Методології, розроблені для розробки IoT відповідно до стандартів ISO/IEC/IEEE; 3.2 Інші пропозиції щодо розробки IoT; 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ; ВИСНОВКИ; СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ; ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Тема роботи; Мета, задачі, Етапи ЖЦ IoT-систем; Типи Agile-методологій для IoT-систем; Принципи гнучкої (Agle) розробки; Стандарти ЖЦ систем та ПЗ; Особливості ЖЦ IoT-систем; MDD та MDE; Багаторівнева архітектура IoT-систем; ВИСНОВКИ

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С, к.т.н., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.11.24-15.11.24	*Виконано*
2.	Підбір наукових джерел по темі роботи	16.11.24-20.11.24	*Виконано*
3.	Переклад та опрацювання наукових джерел	20.11.24-23.11.24	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.11.24-10.12.24	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	11.12.24-12.10.24	*Виконано*
6.	Оформлення другого розділу	12.12.24-13.12.24	*Виконано*
7.	Оформлення третього розділу	13.12.24-14.12.24	*Виконано*
8.	Виконання завдання до підрозділу "Охорона праці"	08.12.24-09.11.24	*Виконано*
9.	Виконання завдання до підрозділу "Безпека в	10.12.24-12.12.24	*Виконано*
	надзвичайних ситуаціях"		
10.	Оформлення кваліфікаційної роботи	12.12.24-31.12.24	*Виконано*
11.	Нормоконтроль	14.12.24-15.12.24	*Виконано*
12.	Перевірка на плагіат	15.12.24	*Виконано*
13.	Попередній захист кваліфікаційної роботи	18.12.24	*Виконано*
14.	Захист кваліфікаційної роботи	27.12.2024	

Студент _____ Щеснюк Т.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Аналіз методологій розробки IoT-систем з використанням Agile-технологій." // Кваліфікаційна робота освітнього рівня "Магістр" // Щеснюк Тетяна Омелянівна // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНмд-61 // Тернопіль, 2024 // с. – 62, рис. – 4, табл. – 2, джерел – 72.

Ключові слова: Agile, Scrum, Kanban, IoT, інформаційна система, розробка

У цій роботі ми розглядаємо основи, включені в маніфест гнучкої (Agile) розробки програмного забезпечення, приділяючи особливу увагу методології Scrum, щоб визначити її роль у розробці інформаційних систем на основі IoT.

Для розробки таких систем, крім Scrum, використовуються також і інші гнучкі методології, такі як eXtreme Programming (XP), Kanban і швидке прототипування. У представленому тут аналізі існуючі методології для розробки IoT були згруповані відповідно до різних підходів, на яких вони базуються, таких як гнучкість, моделювання та спрямування на обслуговування. У цьому дослідженні також аналізується, чи враховують різні пропозиції стандартні етапи процесу розробки чи ні: планування та збір вимог, аналіз рішення, дизайн рішення, кодування рішення та модульне тестування (конструкція), інтеграція та тестування (впровадження), а також експлуатація та обслуговування. Крім того, здфйснено огляд автоматизованих фреймворків, платформ та інструментів, які використовуються в проаналізованих методологіях для покращення розробки систем на основі IoT.

ANNOTATION

"Analysis of IoT Development Methodologies Using Agile Technologies" // Master's degree qualification paper // Shchesniuk Tetiana // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНмз-61 // Ternopil, 2024 // p. – 62, fig. – 4, tables – 1, references – 72.

Key words: Agile, Scrum, Kanban, IoT, information system, development

In this paper, we examine the principles outlined in the Agile Software Development Manifesto, with a particular focus on the Scrum methodology, to determine its role in the development of IoT-based information systems. In addition to Scrum, other Agile methodologies such as eXtreme Programming (XP), Kanban, and rapid prototyping are also employed for the development of such systems. In the analysis presented here, existing IoT development methodologies have been grouped based on the different approaches they rely on, such as agility, modeling, and service orientation. This study also analyzes whether various proposals account for the standard stages of the development process: planning and requirements gathering, solution analysis, solution design, solution coding and unit testing (construction), integration and testing (implementation), as well as operation and maintenance. Furthermore, we include a review of automated frameworks, platforms, and tools used in the analyzed methodologies to enhance the development of IoT-based systems.

ЗМІСТ

ВСТУП

Актуальність задачі.

У наш час з’являється новий тип систем завдяки розвитку та популяризації технологій, пов’язаних з Інтернетом речей (IoT). IoT призначені для моніторингу та контролю середовища за допомогою розгортання датчиків і виконавчих механізмів, які можуть взаємодіяти один з одним і з Інтернет-сервісами. IoT дозволяють нам дистанційно контролювати поточний стан будь-якого фізичного об’єкта чи «речі», змінювати умови навколишнього середовища, отримувати дані для прогнозування чи висновку про події та приймати рішення в режимі реального часу. Для досягнення цих цілей фізичні об’єкти стають цифровими об’єктами, якими можна маніпулювати з будь-якого місця та підключити їх до Інтернету. IoT були застосовані в багатьох сферах, наприклад, для покращення способу життя людей, здоров’я, продуктивності праці, розваг тощо.

IoT – це системи, які змінюють світ і змінюватимуть його ще більше в майбутньому. Однак для повного використання потенціалу IoT потрібні чітко визначені методології розробки, щоб підвищити рівень успішності процесу розробки та якість кінцевої системи.

Зіткнувшись із цією недавньою парадигмою систем, дослідницькій галузі програмної інженерії (Software Engoneering – SE) було доручено запропонувати методології, які можуть конкретно охопити життєвий цикл розробки IoT, оскільки такі системи мають помітні відмінності від традиційних інформаційних систем. Слід зазначити, що IoT складаються не лише з програмних додатків, але мають два додаткових компоненти: апаратне забезпечення (датчики/виконавчі механізми) та механізми зв’язку, які забезпечують взаємодію між цим обладнанням та Інтернетом. Крім того, слід також враховувати взаємодію між речами та людьми через чітко визначені інтерфейси. Таким чином, існує потреба сформулювати та перевірити методології для розробки IoT.

Тому науковцям і розробникам цього нового типу систем потрібна методологія для забезпечення якості їхньої роботи. На сьогоднішній час не існує

загальноприйнятої методології для розробки IoT. Це підтверджується багатьма методологіями розробки IoT, представленими в дослідницьких роботах, знайдених у науково-метричних базах даних, наприклад, сучасні методології розробки IoT [1, 2].

Дослідженнями в області використання гнучких методологій займались також в ТНТУ, в тому числі і в області "розумних міст", що тісно пов'язано з інтернетом речей [3, 4, 5].

Мета роботи.

Метою роботи є поглиблений аналіз найсучасніших методологій розробки IoT, щоб дослідники та розробники могли вибрати найбільш відповідну методологію для розробки IoT.

Для досягнення мети потрібно вирішити наступні задачі:

1. Скласти перелік методологій, що використовувались при розробці IoT систем.
2. Встановити, наскільки методології розробки IoT-систем відповідають стандартам методології розробки IoT, зокрема ISO/IEC/IEEE 15289:2019.
3. Розглянути конкретні аспекти розробки IoT-систем.

Об’єкт дослідження: процеси розробки IoT-систем, як таких, що містять і апаратну, і програмну частину.

Предмет дослідження: інформаційні системи на основі IoT.

Наукова новизна отриманих результатів.

Завдяки аналізу практик та літературних джерел запропоновано поєднання традиційних методологій розробки апаратного забезпечення з гнучкими методологіями створення програмних продуктів в рамках інтегрованої IoT-системи.

Практичне значення отриманих результатів.

Представлено поглиблений аналіз найсучасніших методологій розробки IoT, щоби можна було вибрати найбільш відповідну методологію для розробки конкретної IoT. Крім того, можна використати даний аналіз, щоб визначити

методологію, спеціально створену для розробки IoT, яка відповідає стандартам ISO/IEC/IEEE 15289:2019 і охоплює всі аспекти життєвого циклу таких систем.

Апробація результатів та особистий внесок здобувача.

Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції "Актуальні задачі сучасних технологій – 2024", яка проводилась у ТНТУ.

1 ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ

1.1 Огляд стану проблеми

Першою методологією розробки програмного забезпечення була водоспадна методологія. Ця методологія зазвичай використовується для розробки, виробництва або створення будь-якого фізичного продукту, з особливостями, що оригінальна водоспадна методологія передбачає, що розробник може повернутися до будь-яких попередніх етапів, коли це необхідно, навіть якщо це може бути складним або нездійсненним у багатьох випадках. Навпаки, методології побудови обладнання не розглядають таку можливість, оскільки це призведе до великого негативного економічного впливу.

Враховуючи затримки та обмежені досягнення у впровадженні методології водоспаду, необхідно визначити причини її низького впливу на галузь програмного забезпечення. Однією з таких причин є те, що водоспадна методологія охоплює розробку великих інформаційних систем у цілому [6]. Навпаки, інші, такі як спіральна методологія, допомагають виявити, коли неможливо розробити заплановану систему і, як наслідок, відмовитися від її розробки до того, як інвестувати в неї ресурси [7].

Інша причина полягає в тому, що на стадії аналізу програмної системи зазвичай працюють кінцеві користувачі (клієнти), які не мають дуже чіткого уявлення про функціональні можливості та характеристики якості, котрі необхідно охопити. Таким чином, методологія прототипування [8] виникла для формалізації представлення ітераційних версій продукту кінцевим користувачам для їх оцінки. Прототипування в даний час використовується як частина інших методологій, особливо гнучких методологій.

Гнучкі методології з’явилися для зменшення ризику незавершення розробки великих систем через бюджетні, технологічні чи ресурсні обмеження, серед інших причин. Поява гнучких методологій дозволила підвищити рівень

успішності проектів розробки програмного забезпечення та зменшити споживання бюджету проектів [9].

Гнучкі методології поділяють загальну систему на результати (модулі або підсистеми), щоб замовник використовував або перевіряв ці результати до завершення всієї системи. Гнучкі методології базуються на 4 цінностях і 12 принципах, включених до Маніфесту гнучкої розробки програмного забезпечення [6]. Ці методології зосереджені лише на розробці частини загальної системи, розгортанні повністю функціональних продуктів для цієї частини системи за короткий період часу (від одного до максимум від 4 – 6 тижнів, типово – 2 тижні).

Крім того, деякими характеристиками гнучких методологій розробки програмного забезпечення, на відміну від традиційних методологій, є: команди повинні бути невеликими, результати, які потрібно розробити, повинні бути узгоджені з клієнтом, а зміни можуть бути внесені в проект у будь-який час. Однак у деяких випадках ці особливості можуть перетворитися на недоліки. Наприклад, у проекті розробки корпоративного програмного забезпечення невелика команда розробників, як це запропоновано гнучкими методологіями (у Scrum, 8 розробників), недостатня в умовах жорстких часових обмежень. Крім того, щоразу, коли потрібно внести несподівані зміни в програмне забезпечення, можуть виникнути проблеми з бюджетом, оскільки це може бути вирішено командою розробників, використовуючи початковий бюджет [10]. Більше того, якщо рішення щодо пріоритету розробки кожного продукту в першу чергу залежить від замовника, це може занчно зменшити продуктивність команди розробників.

Що стосується IoT, у наведеному тут аналізі ми виявили, що автори зазвичай завершували свої розробки, використовуючи спеціальну методологію або взагалі без будь-якої чітко зазначеної методології. Одним із прикладів є робота [11], який представляє систему, розроблену для інтеграції існуючих сенсорних мереж і інтелектуального зовнішнього додатку, створеного за сучасними технологіями, але без згадки використаної методології розробки.

Іншим прикладом є робота [12], яка описує кілька IoT, розроблених для різних цілей, включаючи охорону здоров’я, сільське господарство та туристичну галузь. Проте процес дослідження показує, що в розробці не було дотримано певної методології. Крім того, деякі автори розробили IoT за методологіями, які не створені спеціально для цього типу систем. Парою прикладів цього є поєднання Scrum з XP [13] і поєднання Scrum з RP [14].

1.2 Сучасні методи розробки IoT-систем

SE відповідає за забезпечення адекватної методології для розробки всіх типів комп’ютерних систем. Тому дослідники в цій галузі працювали над тим, щоб забезпечити оптимальні методології для розробки різних типів систем, які з'являються разом з розвитком технологій та інструментів програмування.

У літературі існує кілька методологій розробки IoT. Наприклад, INTER-METH, який був представлений у [15]. Це методологія для розробки IoT, заснована на водоспадній моделі, що робить її ітераційною. Аналогічно, методологія розробки на основі тестування для IoT (TDDM4IoT), представлена у [16], базується на маніфесті гнучкої розробки програмного забезпечення та відповідає етапам життєвого циклу розробки системи. Однак жодна з них не є загальновизнаною. Цей факт є очевидним у оглядах найсучасніших методологій розробки IoT, які містяться в розглянутих джерелах з наукометричних баз даних, які представлені нижче.

У роботі [17] автори представляють сучасний огляд методологій розробки IoT, які застосовуються до розумних світлофорів. Вони вважають, що існує обмежена кількість методологій, які можна використовувати для розробки цих типів IoT і чітко представляють їхні специфічні характеристики. Автори надають інформацію для підтримки прийняття рішень невеликою групою розробників, коли вони вирішують, якій методології слідувати.

Автори в [18] також розглянули аналіз існуючих методологій для розробки IoT. Однак продукти, проаналізовані в дослідженні, базуються на опитуваннях

третіх сторін, а не на наукових публікаціях. Деякі аспекти, проаналізовані в переглянутих методологіях, збігаються з аспектами цього дослідження, наприклад, етапи життєвого циклу розробки. Щоб уніфікувати термінологію, знайдену під час аналізу (методологія, структура, платформа, інструмент), автори [18] брали до уваги стандарт ISO/IEC/IEEE 24765, SEBoK (Сукупність знань системної інженерії) і PMBoK (Сукупність знань з управління проєктами). Однак жодна з попередніх робіт із огляду найсучаснішого стану не згадує стандарти ISO/IEC/IEEE, на яких базується аналіз.

Це дослідження є актуальним через те, що в літературі не знайдено вичерпного огляду найсучасніших IoT, щоб зробити висновок про те, чи існує універсально прийнята методологія для розробки таких систем. Ця робота також має на меті представити огляд мінімальних умов виключення та ретельний аналіз існуючої літератури. Крім того в даній роботі пропонується виділити і методології, якщо вони відповідають міжнародним стандартам, щоб гарантувати якість розроблених IoT.

Досі методологія розробки IoT не була встановлена в ISO/IEC/IEEE 15289:2019 як стандарт, який уніфікує процеси, визначені в ISO/IEC/IEEE 12207:2017 та ISO/IEC/IEEE 15288: 2015. Хоча жодна із запропонованих методологій не охоплює всі технічні аспекти життєвого циклу програмних систем, викладених у ISO/IEC/IEEE 15289:2019, висновки з існуючої літератури свідчать про те, що TDDM4IoT [16] є єдиною методологією, життєвий цикл розробки якої IoT відповідає такому стандарту.

Методологію розробки системи можна визначити як «серію етапів створення програмного або апаратного забезпечення за шаблоном, заснованим на досвіді та теорії проектування програм». З іншого боку, можливе визначення методології розробки програмного забезпечення було б «процесом поділу роботи з розробки програмного забезпечення на менші, паралельні або послідовні кроки або підпроцеси для покращення дизайну та управління продуктом» [19]. Таким чином, і системи, і методологія розробки програмного забезпечення можуть включати попереднє визначення конкретних результатів і

артефактів, які команда проекту створює та завершує для розробки або підтримки програми або системи.

Розробка IoT з використанням методологій, розроблених для розробки традиційних інформаційних систем, має великий недолік, оскільки вони не охоплюють конкретних аспектів IoT, таких як проектування та розгортання апаратного забезпечення (наприклад, датчиків, виконавчих механізмів, процесорів тощо) у середовище, яке потрібно контролювати. Крім того, інші аспекти, які хоча і не є унікальними для IoT, є важливими для системи цього типу. Це включення методів штучного інтелекту (ШІ), щоб допомогти системі в прийнятті рішень і бути контекстно-зрозумілою, тобто вона реагує належним чином відповідно до контексту або існуючих умов у середовищі. Тому, якщо ми також врахуємо неоднорідність компонентів і сфер застосування IoT, очевидно зробити висновок, що професіонали з різних областей повинні бути частиною команди розробників, щоб виконувати діяльність, включену в різні стадії розробки, діяти протягом усього життєвого циклу або виконувати певні завдання.

2ЕТАПИ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ IoT-СИСТЕМ

Щоб провести аналіз методології розробки системи, ми могли б повернутися до 50-х і 60-х років, коли з'явилося багато мов програмування, таких як Fortran, Cobol або Basic. Перші методології розробки програмного забезпечення для настільних систем були визначені відповідно до парадигми, на якій базувалася мова програмування для реалізації системи. Наприклад, для розробки з використанням структурованих мов програмування з'явилися структуровані методології розробки [20], тоді як для розробки з використанням об'єктно-орієнтованих мов програмування з'явилися об'єктно-орієнтовані методології [21]. Згодом методології були переорієнтовані відповідно до типу системи, яка буде розроблена, наприклад настільна чи веб.

Водоспадна методологія включала наступні фази або етапи: (a) аналіз вимог до системи, (b) аналіз вимог до програмного забезпечення, (c) попередній дизайн програми, (d) аналіз системи, (e) дизайн програмного забезпечення, (f) кодування, (g)) тестування та (h) експлуатація.

Згодом гнучкі методології, серед яких популярні XP, Kanban і Scrum [22], вважають, що цінності та принципи гнучкого маніфесту повинні бути присутніми в методології розробки будь-якого програмного продукту, який можна розбити та розробляти частинами. При розробці будь-якого IoT-пристрою та системи необхідно враховувати: розгортання різних апаратних компонентів (наприклад, датчиків, виконавчих механізмів, процесорів тощо) у середовищі, яким потрібно керувати, зв'язок та взаємодію між пов'язаними об'єктами чи «речами», і розробка засобів для взаємодії користувача з системою (наприклад, веб-додаток, мобільний додаток тощо), серед інших аспектів, які з часом стануть результати. Ці результати будуть розроблені шляхом виконання різних заходів. Отже, ми можемо зробити висновок, що цей тип системи можна розробити за допомогою гнучких методологій.

2.1 Стандарти, що визначають етапи та процеси життєвого циклу програмного забезпечення

ISO/IEC 12207:2017 стосується процесів життєвого циклу програмного забезпечення, тоді як ISO/IEC/IEEE 15288:2015 стосується процесів життєвого циклу системи. У свою чергу, стандарт ISO/IEC/IEEE 15289:2019 пропонує загальний життєвий цикл для розробки систем і програмного забезпечення на основі процесів життєвого циклу, визначених у двох стандартах. Так само метою ISO/IEC/IEEE 24748-1:2018 є сприяння спільному використанню вмісту ISO/IEC/IEEE 15288 та ISO/IEC/IEEE 12207, надаючи уніфіковані та консолідовані настанови щодо систем та управління життєвим циклом програмного забезпечення. З іншого боку, ISO/IEC/IEEE 24748-3:2020 містить настанови щодо застосування стандарту процесу життєвого циклу програмного забезпечення, тобто ISO/IEC/IEEE 12207:2017. Крім того, стандарт ISO/IEC/IEEE 24748-4:2016 містить детальні вимоги та настанови щодо застосування процесів життєвого циклу системи, тобто ISO/IEC/IEEE 15288. Рисунок 2.1 було підготовлено після перегляду стандартів, щоб показати, життєві цикли та процеси, які включають ці стандарти.

Методології можуть вільно вказувати, як виконувати етапи та технічні процеси, запропоновані стандартами ISO/IEC/IEEE, порядок їх виконання та навіть вибирати, які процеси виконуватимуться. Проте, ймовірно, щонайменше такі процеси повинні бути присутніми в життєвому циклі:

1) планування та збір вимог;

2) аналіз рішення;

3) проектування рішення;

4) кодування рішення та модульне тестування (конструювання);

5) інтеграція та тестування (впровадження);

6) експлуатація та обслуговування.

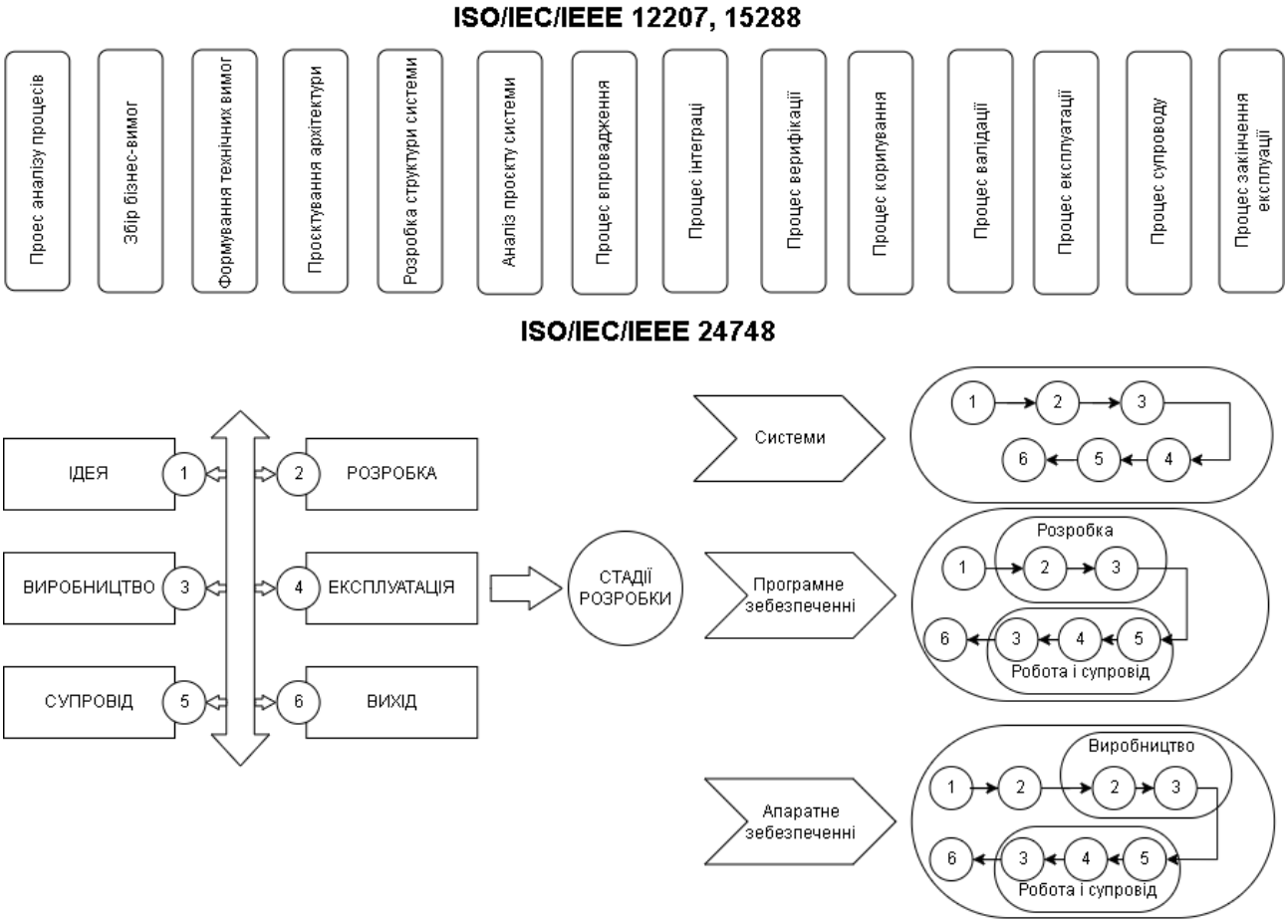

Рисунок 2.1 – Підсумок етапів і процесів життєвого циклу програмних систем, розглянутих у різних розглянутих стандартах ISO/IEC/IEEE 12207, 15288, 24748

Крім того, уся робота, виконана в кожному з проведених заходів, повинна бути добре задокументована. Таким чином, різниця між методологіями має полягати в процесах, які слід розглядати, у способі їх виконання та порядку їх виконання.

2.2 Методології розробки IoT на основі Agile Manifesto

Ці методології розбивають систему, яку потрібно розробити, на готові продукти, які, у свою чергу, організовані в завдання, які тривають від 2 до 4 тижнів (так звані спринти у Scrum). Такий спосіб організації роботи контрастує

з іншими методологіями розробки, такими як водоспад, спіраль, RP тощо, які вирішують проблему повністю.

2.2.1 Інструкції з управління ризиками проекту

Гнучкі методології розглядають ризики опосередковано, встановлюючи характеристики робочих команд. Так, наприклад, у [23] представляють методологію для малих команд, засновану на принципах, визначених у Scrum [24]. Таким чином вони намагаються компенсувати відсутність чітких вказівок щодо маніфесту гнучкої розробки програмного забезпечення, щоб визначити етапи або дії, пов’язані з ризиками під час розробки систем.

Автори вважають, що створення невеликих команд розробників і проведення щоденних особистих зустрічей є двома характеристиками, які можуть зменшити ризики, властиві залученим людям. Поступове надання кінцевих результатів також може зменшити ризики, властиві самій технології (її відсутність на ринку, ступінь домінування розробників тощо), і бюджету (придбання компонентів, навчання персоналу тощо).

Agile-маніфест (і, відповідно, Scrum) нічого не визначає щодо процесів розробки програмного забезпечення. Отже, Scrum можна розглядати як методологію управління проектами, а не як методологію розробки програмного забезпечення, оскільки вона не визначає дії, які повинні бути виконані протягом життєвого циклу розробки, у той час як вона визначає, як керувати життєвим циклом проекту [25]. Автори [26, 28] включають аналіз ризиків як невід’ємну частину кожного спринту в методології Scrum.

У гнучких методологіях, на які спрямована ця робота, зазначені процеси виконуватимуться більше одного разу, залежно від кількості результатів, на які розділено систему. Порядок, у якому вони виконуються, може бути різним, залежно від конструкції методології. Наприклад, тести в Scrum виконуються в кінці [24], тоді як в XP вони пишуться на початку циклу розробки, як це передбачено TDD (Test-Driven Development) [27].

2.2.2 Процес збору вимог кінцевого користувача

Однією з причин успіху гнучких методологій є відмінна риса можливості вносити зміни під час розробки системи на будь-якому етапі. В ідеалі всі вимоги повинні бути відомі і зрозумілі з самого початку розробки, хоча це не завжди так. Виявлення вимог є проблемою, відомою розробникам [29], яка змушує розробників застосовувати зміни, запропоновані клієнтами або кінцевими користувачами, протягом усього процесу розробки.

Для гнучких методологій стандарт ISO/IEC/IEEE 26515:2018 визначає історії користувачів, сценарії та персонажів як інструменти для отримання та аналізу вимог, тоді як варіанти використання пропонуються як техніка проектування. Наприклад, Scrum пропонує використовувати історії користувачів як інструмент для отримання вимог [24]. Історії користувачів – це прості наративи, які ілюструють вимоги користувача з точки зору людини чи актора. Історії користувачів повинні бути повністю зрозумілі розробникам, щоб висловити вимоги до системи та програмного забезпечення. Історії користувачів написані природною мовою. Тому вони є неструктурованими інструментами і можуть бути неправильно витлумачені розробниками.

Важливо, щоб серед розробників був консенсус щодо правильного отримання системних вимог, оскільки вони необхідні для визначення початкової архітектури системи. Отримання вимог складне, і навряд чи вдасться отримати всі на початку розробки системи. Таким чином, у цьому сценарії можна передбачити, що будуть відбуватися зміни протягом розробки системи. Безсумнівно, можливість адаптації до таких змін, як це можуть зробити гнучкі методології, буде важливим моментом для включення в керівні принципи методології, спеціально розробленої для розробки IoT.

2.2.3 Нефункціональні вимоги

IoT дуже важливо враховувати нефункціональні вимоги (Non Functional Requirements – NFR). Безпека та конфіденційність даних, отриманих датчиками,

довговічність джерел живлення (батареї), стійкість комунікацій і неінтрузивність датчиків, серед інших NFR, необхідно враховувати на кожному етапі IoT розробки. Тому, наприклад, існують напрямки дослідження щодо різних методів, протоколів і вказівок для гарантування безпеки та конфіденційності даних [30], низького енергоспоживання або для інтеграції датчиків з різними рівнями втручання серед інших.

Отже, важливість, яку надають NFR в IoT, може зробити гнучкі методології невідповідними для їх розробки, оскільки вимоги виявляються через історії користувачів через проблеми, згадані в попередньому підпункті. Історії користувачів можна використовувати для ідентифікації NFR, коли вони є потребами користувачів, але, згідно [31], Scrum не має чіткого способу перевірити, чи виконуються NFR чи ні. Цей ризик можна зменшити або навіть уникнути його повністю, якщо критерії прийнятності визначено детально, а NFR чітко описані в історіях користувачів із самого початку, покладаючись на додаткову інформацію та подальші дії з цією метою. Щоб забезпечити успіх в отриманні вимог, написання історій користувачів (якщо такий інструмент використовується для їх отримання) має здійснюватися обома сторонами, тобто розробниками та кінцевими користувачами, які також мають бути залучені до розробки IoT, щоб переконатися, що це нарешті задовольнить їхні потреби та очікування.

2.2.4 Кількість членів команди розробки

PMBoK [25] надає вказівки для управління робочими групами від малих до великих. Однак вважається, що популярні гнучкі методології працюють з невеликими командами максимум з 15 осіб, включаючи власника продукту. Наприклад, Kanban вказує максимум 14 розробників, XP і Scrum, і максимум на 11 учасників, разом із scrum-майстром і власником продукту [32]. Стосовно ролей команди в Scrum, то в [31] автори підкреслюють необхідність створення інших ролей, крім тих, які визначені в Scrum для великих програмних компаній.

У цьому ж ключі в роботі [33] також підкреслюють необхідність адаптації Scrum для великих програмних проектів.

Насправді вони використовують техніку Scrum of Scrums (SoS) і додають дві нові ролі, які називаються Генеральним власником продукту (GPO) і Генеральним Scrum Master (GSM). Ці нові ролі слугуватимуть для координації та допомоги іншим ролям.

2.3 Моделювання як ключ у методології розробки IoT

З точки зору розробників програмного забезпечення, IoT в основному характеризуються неоднорідністю своїх компонентів і технологій, які вони включають, на додаток до обмеженої потужності обробки кожного компонента. Наприклад, через відсутність встановленим стандартам на апаратному рівні виробники часто пропонують різні реалізації та/або робочі функції, що часто призводить до гетерогенного програмного забезпечення та комунікаційної платформи (тобто з різними технологіями та протоколами зв’язку). Крім того, розробники зазвичай повинні розгортати спільний набір функціональних можливостей на кількох різних пристроях. Отже, у цьому сценарії неоднорідність апаратного забезпечення призводить до різних вихідних кодів програмного забезпечення не зважаючи на ідентичний дизайн, щоб мати можливість підтримувати різні функції основного апаратного забезпечення.

Методології розробки на основі моделі можуть допомогти вирішити неоднорідність апаратного забезпечення. Насправді ці методології були введені з метою зосередження уваги на проектуванні функціональних можливостей системи, головним чином на незалежному від платформи рівні абстракції. Її головна мета полягає в тому, щоб отримати остаточну реалізацію системи з якомога меншою кількістю залежного від платформи коду, написаного розробниками.

Найважливішими методологіями розробки на основі моделі, відомими на даний момент, є інженерія на основі моделі (Model-Based Engineering – MBE)

[34], інженерія, керована моделлю (Model-Driven Engineering – MDE) [35], розробка, керована моделлю (Model Driven Development – MDD) [35], і архітектура, керована моделлю (Model-Driven Architecture – MDA) [35]. Хоча подібність між ними легко розпізнати, їх відмінність полягає у важливості, яку вони надають самим моделям, у тому, як вони концептуально визначають, «що таке» модель, і в тому, як вони використовуються, щоб остаточно отримати реалізацію системи. Крім того, вони відрізняються етапами розробки та інструментами, які вони пропонують для забезпечення моделювання.

І MDD, і MDA є рекомендаціями, яких слід дотримуватися на етапах розробки програмного забезпечення.

Різниця між ними полягає в тому, що MDD можна написати на будь-якій мові моделювання, тоді як MDA є стандартною специфікацією, яка пояснює, що UML (Unified Modeling Language) має використовуватися, як основна мова моделювання, тоді як будь-яке перетворення має вказуватися за допомогою Query/View/Transformation (QVT) мови. MDA також уточнює, що моделі повинні бути перетворені в порядку спадання рівня абстракції, тобто від обчислювально незалежних моделей до незалежних від платформи моделей. Як зазначено в стандартній специфікації, MDA – це підхід до проектування, розробки та реалізації програмного забезпечення, який очолює Група управління об’єктами (Object Management Group – OMG). MDA надає вказівки щодо структурування специфікацій програмного забезпечення, які виражаються як моделі. Таким чином, MDA має на меті залишити осторонь технічні особливості реалізації, а натомість зосереджується на «моделюванні» програмних рішень. З іншого боку, він відокремлюється від виявлення вимог, припускаючи, що це попередня робота, яка вже завершена на більш ранньому етапі.

У MBE ані визначення моделі, ані автоматична генерація коду не є ключовими аспектами процесу розробки. Крім того, вона розглядає моделі як плани, які повинні бути зрозумілі програмістам для написання програмного коду цільовою мовою програмування. Натомість MDE керується моделями, оскільки очікується, що моделі будуть визначені для (напівавтоматичної) генерації

принаймні однієї часткової кодової бази або навіть інших моделей з них [36]. Цей процес зазвичай називають перетворенням з моделі на текст (M2T) або з моделі на модель (M2M). Крім того, у MDE моделі та перетворення можуть бути визначені будь-якою мовою моделювання та можуть посилатися на різні рівні абстракції.

2.4 Методології традиційної розробки, застосовані до розробки IoT

Як зазначалося вище, розроблені методології використовувалися для розробки звичайних інформаційних систем, деякі з них були адаптовані для розробки IoT. Щоб переглянути літературу з цього питання переглянули деякі науковометричні бази даних, а саме Web of Science (WoS), Scopus, IEEE та ACM, щоб переконатися, що ми розглядаємо найбільшу кількість статей, опублікованих у цьому дослідженні тема.

У роботах, представлених на рисунку 2.2, також визначено області або сфери застосування, в яких запропоновані IoT були розроблені. Результати цього аналізу можна побачити на рисунку 2.3, де видно, що сфера, в якій розробка IoT була найбільш формалізована, це охорона здоров'я з 18,42%, за якою слідують розумні автомобілі та якість повітря з 10,53. % у кожному з них по відношенню до загальної кількості робіт, розглянутих для цього аналізу.

Хоча зростання IoT в останні роки було стрімким, не всі ще мають доступ до Інтернету. За даними Світового банку, лише 49,72% населення має доступ до Інтернету. Насправді є країни, населення яких має доступ до Інтернету менше 25%. Імовірно, ще менший відсоток людей знає, що таке IoT і яку користь IoT може їм принести. Одна з методологій, яка була успішною, коли клієнт не зрозумів своїх вимог або через те, що вони не знають, як можна використати існуючу технологію, або тому, що вони не дуже чітко розуміють вимоги системи, яку потрібно розробити., це методологія прототипування. Таким чином, ці дані свідчать про те, що використання прототипування для розробки такого типу системи буде успішним і буде добре оцінено клієнтами.

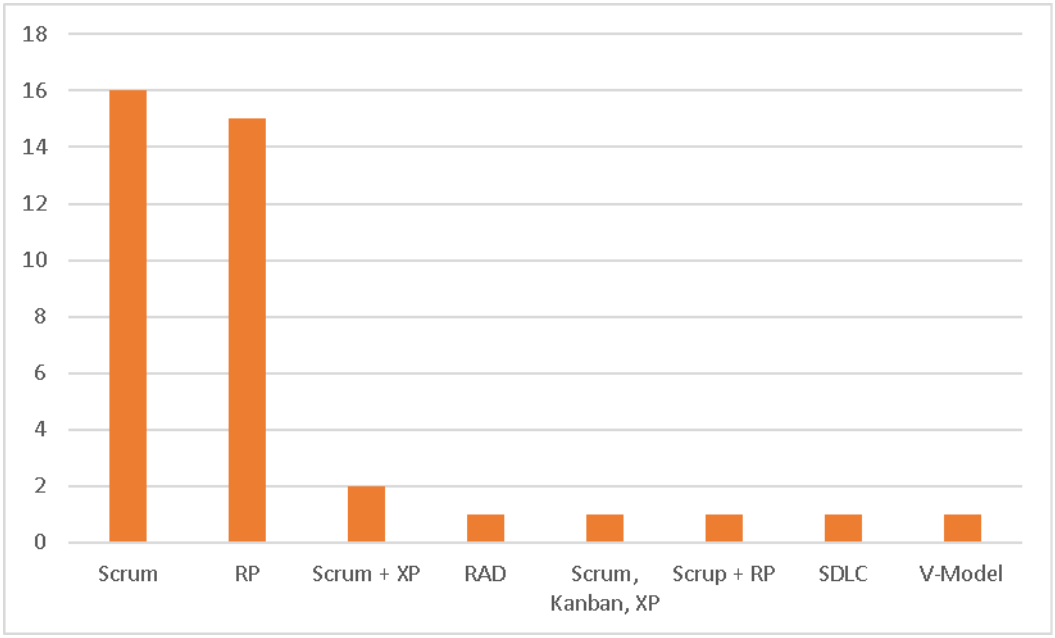

Рисунок 2.2 – Кількість IoT, розроблених за методологіями, призначеними для розробки інформаційних систем

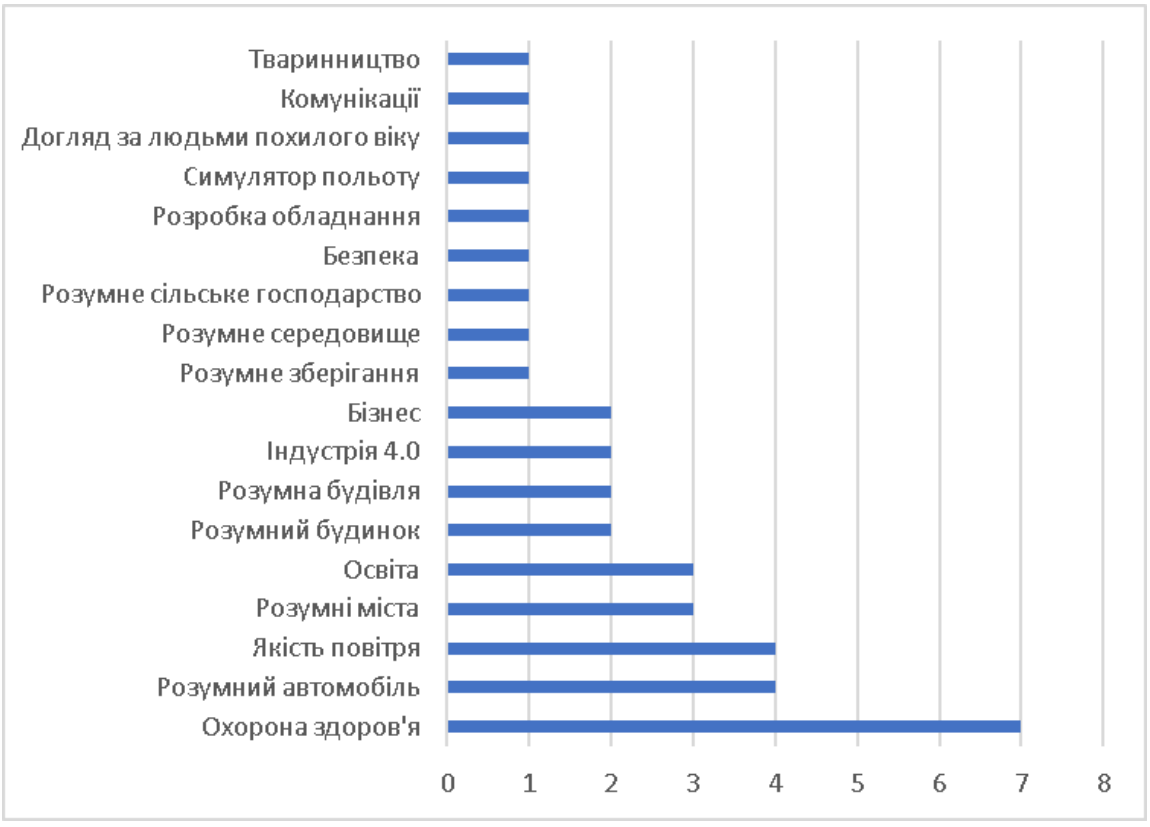

Рисунок 2.3 – Сфери застосування IoT розроблені за допомогою методологій, розроблених для ІС

Крім того, можливість не продовжувати розробку системи через брак бюджету, технологічні проблеми (наприклад, недоступність технології) або будь-якого іншого характеру є ще однією характеристикою, яка підтримує успіх гнучкої розробки.

ЗМЕТОДОЛОГІЇ, ІНСТРУМЕНТИ ТА ФРЕЙМВОРКИ, ОРІЄНТОВАНІ НА ПРОЕКТУВАННЯ ТА СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ IoT

Для створення програмного забезпечення представлені технології, здатні автоматично генерувати код на різних мовах. Більшість із них зосереджено на створенні коду на C, C++ або їх варіантах, оскільки вони є найпопулярнішим типом мов на контролерах, які використовуються при розробці IoT. Ще одна мова програмування, на якій вони генерують код, – Java, і лише в одному зі знайдених посилань згадується генерація коду для середовища Node.js.

Щоб розпочати етапи проектування та створення програмного забезпечення, потрібно спочатку пройти етапи аналізу потреб зацікавлених сторін і виявлення системних вимог, а потім перейти до етапу аналізу системних вимог і вимог до програмного забезпечення. Ці етапи розглядаються як дуже важливі етапи, оскільки вони є свого роду постачальники інформації, необхідної для продовження проектування та розробки. Однак інші роботи вважають їх вирішеними, приділяючи їм менший акцент, ніж у попередніх роботах, і не вказуючи жодного методу аналізу чи інструментів, які слід використовувати.

Внесок [37] полягає у розробці потоку проектування IoT на основі MDE і SOA. Ці автори зосереджуються на моделях бездротової мережі IoT (WPAN). Ця пропозиція також підтримує моделювання та реалізацію функцій додатків для їх розгортання в системі IoT.

Діяльність з проектування підтримується процесами верифікації та підтвердження вимог, що полегшує вдосконалення моделі системи. Це забезпечує відповідність NFR, пов’язаним із продуктивністю та ефективністю додатків, а також функціональним вимогам (FR). Їхня робота зосереджена на платформі, яка використовує власну мову DSL (Domain Specific Language), яка служить для написання служб REST, що працюють на цій платформі.

MDE4IoT – це методологія, заснована на MDE, яка зосереджена на моделюванні та створенні кінцевого продукту. У цій методології не згадуються

етапи планування, отримання та аналізу вимог, експлуатації, обслуговування або розгортання. Виявлення системних вимог також не розглядається його авторами. Щоб досягти трансформації моделей у виконувані артефакти, MDE4IoT використовує комбінацію доменно-орієнтованих мов моделювання (DSML). Моделювання виконується з трьох точок зору: (1) конкретної області застосування програмного забезпечення, (2) фізичних пристроїв і (3) як програмного, так і апаратного забезпечення конкретної області застосування.

IOPT-Tools дозволяє моделювати керування для вбудованих систем за допомогою класу мереж Петрі, які називаються «Мережі входу-виводу місця-переходу». IOPT працює через веб-інтерфейс і дозволяє отримати керуючий код з однієї графічної моделі. Однак, якщо ви хочете обмінюватися даними між різними контролерами, ви повинні вручну написати та адаптувати згенерований код. Тому, хоча це інструмент для пристроїв IoT загалом, він не розглядає, як реалізувати взаємодію з іншими компонентами IoT або з самими користувачами.

У роботі [38] представляють підхід для створення мобільних додатків для IoT. Цей підхід базується на розширенні UML, відомому як Interaction Flow Modeling Language (IFML), призначеному для вираження вмісту, взаємодії з користувачем і керування поведінкою інтерфейсу мобільних додатків. Для моделювання як подій, так і дій, пов’язаних із пристроями IoT, до цього набору графічних нотацій (IFML) додано нові елементи для створення моделей, які візуально представляють поведінку систем під час взаємодії користувачів. Щоб визначити шаблони, які охоплюють найпоширеніші випадки використання IoT, автори визначають моделі класів вмісту та моделі класів взаємодії. Розглянуті шаблони стосуються IoT, взаємодії з користувачем і синхронізації даних. На яких мовах програмування генерується програмний код, не уточнюється. Крім того, ця пропозиція охоплює деякі обмежені сфери застосування, і навіть у цих областях вона обмежена лише певними типами систем.

Автори у [39] пропонують підхід MDE, який дозволяє розробникам отримувати проект системи із загальної специфікації своїх вимог. Їхня методологія проектування дотримується парадигми «зверху вниз», і вони

роблять ставку на автоматичні процеси для визначення поведінки глобальних вимог системи до набору компонентів для спільної роботи для усунення можливих помилок. Кожен рівень абстракції описується за допомогою певної метамоделі. Таким чином, застосування підходу MDE вимагає визначення відповідних метамоделей і відповідних перетворень моделі.

Є пропозиція подавати структуру розробки, яка охоплює наступне: специфікація домену, специфікація дизайну архітектури програми, генерація архітектури, генерація домену, визначення набору абстрактних взаємодій з користувачем, генерація інтерфейсів користувача та опис специфікації реалізації. Іншим його міркуванням є генерація коду програм, які можна розгортати на самих пристроях. Для кожного з цих аспектів методології визначається мова, яка використовуватиметься різними членами групи розробників.

ROOD (Resource-Oriented and Ontology-Driven Development) – це методологія, заснована на підході MDA, хоча вона підтримується інструментами на основі MDE. ROOD орієнтований на розвиток інтелектуальних просторів з двох точок зору: (1) контекстної діяльності або поведінки ресурсів, які є датчиками та приводами, і (2) інтелектуального об’єкта. Він включає моделі контексту середовища (ECM) і моделі смарт-об’єктів (SOM). Серед інструментів на основі MDE ROOD містить профіль UML, відомий як мова моделювання інтелектуального простору (SsML). Ця методологія представляє 3 основні етапи, а саме перший етап, ECM, який пов’язаний з MDA CIM. Другий етап, SOM, представлений MDA PIM і, нарешті, PSM. На кожному етапі розглядають перевірку узгодженості моделі та семантичної узгодженості з точки зору ECM або SOM з відповідними точками зору та відповідно до концепцій домену в базі знань. Хоча ROOD вважається методологією, яка не відповідає стандартам ISO, вона чітко розкриває роботу, яку кожен професіонал повинен виконувати в процесі розробки, включаючи аналіз, моделювання та впровадження.

Нарешті, ELDAMeth [40] – це методологія на основі моделювання для розподілених агентних систем (DAS), яка дозволяє швидко створювати

прототипи на основі візуального програмування, перевірки та автоматичної генерації коду для DAS на основі фреймворку Java Agent Development (JADE). ELDAMeth – це ітеративний процес розробки для DAS, який охоплює етапи низькорівневого проектування, перевірки на основі моделювання та реалізації, орієнтованої на JADE. Однак тут не представлено інформацію про те, як автори підходять до етапів планування, виявлення та аналізу вимог, на додаток до етапів інтеграції, експлуатації та обслуговування.

Усі методології розробки, представлені в цьому розділі, зосереджені на етапах проектування та побудови (автоматична генерація коду) на основі підходу трансформації моделі для отримання програмного коду IoT. Проте деякі з них базуються саме на трансформації моделей, тоді як інші базуються на патернах, а інші – на агентах.

3.1 Методології, розроблені для розробки IoT відповідно до стандартів ISO/IEC/IEEE

Було виявлено декілька наукових публікацій, які представляють певну методологію розробки для IoT. Під час швидкого перегляду відібраних документів можна було визначити, що більшість із них відповідає менше ніж на 50% етапам, установленим стандартами ISO/IEC/IEEE у життєвому циклі систем/програмного забезпечення. З цієї причини ми переглянули літературу, щоб визначити, які стадії повинні мати відповідний життєвий цикл і чи існує консенсус з цього питання. Загальні етапи життєвого циклу систем: (1) планування, (2) аналіз (вимог і програмного забезпечення/системи), (3) проектування рішення, (4) кодування рішення та тестування (створення), (5) інтеграція та тестування (впровадження), і (6) експлуатація та обслуговування. Таким чином, ми вважали, що ці етапи є тими, які методологія повинна визначити.

Аналіз документів, що представляють ці методології, дозволив визначити ступінь відповідності (виражений з достатньою чіткістю у відповідному документі) цих 6 етапів, оцінка яких була наступною:

– Належна відповідність. Автори розглянули всі етапи життєвого циклу програмного забезпечення/системи, записавши докази цього, наприклад, пояснюючи, з чого він складається, та інструменти, які будуть використовуватися, серед інших аспектів.

– Неповна відповідність. Хоча автори конкретно не називають деякі етапи, називають деякі види діяльності, інструменти, які необхідно використовувати, або інші аспекти цих відсутніх етапів. Наприклад, вони згадують використання діаграм варіантів використання, діаграм класів або створення програмного забезпечення з моделей, серед іншого.

– Відповідність стандартам. Автори прямо не називають деякі етапи, оскільки вони є частиною або вже присутні в підходах, на яких базуються. Наприклад, у деяких випадках вони згадують, що базуються на основах SCRUM, що означає, що етап планування виконується.

– Неадекватна відповідність. Твори, в яких згадується важливість певного етапу, але не наводиться докладніша інформація про те, як його виконати.

– Відсутня відповідність. Роботи, в яких автори не згадують діяльність окремих етапів або навіть не згадують певні етапи.

RASPSS [41] – це методологія проектування та впровадження (конструювання) інтелектуальних процесів охорони здоров’я в контексті промислового взаємозв’язку та ітеративного проектування пристроїв допомоги в реабілітації. Ця методологія, крім того, що спрямована на певний тип системи, веде безпосередньо до етапу побудови цього типу системи. Представлена в ньому архітектура чітко демонструє важливість взаємодії з користувачем. У ньому детально розглядається створення пристроїв IoT, включаючи забезпечення їх техніками штучного інтелекту, необхідними для досягнення поставлених цілей.

Робота [42] зосереджена на розробці атомарних послуг. Сервіси розгортаються в репозиторії сервісів відповідно до типів доступних обчислювальних ресурсів. Цей репозиторій був розроблений для надання двох функціональних можливостей: (1) перелік послуг із відповідними описами; і (2) підключаються образи Docker, готові до використання в різних архітектурах на основі сервісів. Його архітектура базується на сховищі форм образів Docker і контролюється процесом оркестровки служби. Служби розробляються за життєвим циклом із етапами проектування, розробки (стадія цієї методології), розгортання та виконання. Крім того, IoT розглядаються як складні системи та виконуються через композицію послуг. Третє питання стосується компонентів, пов'язаних з обчислювальними ресурсами (обробка, зберігання тощо) та комунікаціями (протоколи, технології тощо). Ця методологія орієнтована на етап побудови IoT і закрита для технологій. Це може бути причиною, чому етапи планування, інтеграції та обслуговування не згадуються. Передбачається, що інші дії відбуваються на тому, що вони називають стадією розробки.

TDDM4IoT [16] – це методологія, заснована на 4 цінностях і 12 принципах Agile Маніфесту. Вона розглядає 11 етапів і визначає прийнятним чином ресурси та інструменти, які будуть використовуватися на кожному з етапів. Найбільше пов'язана із життєвим циклом програмного забезпечення/системи, викладеним у стандартах ISO/IEC/IEEE. TDDM4IoT підвищує етапи в особливий спосіб, залучаючи ті аспекти IoT, які відрізняють їх від традиційних інформаційних систем, за винятком того, що не стосується тих дій, які передбачають надання технологій ШІ для IoT.

Ще одна методологія, заснована на принципах і цінностях гнучких методологій, представлена у [43]. Ця методологія зосереджена на створенні IoT. Вона представляє етап виявлення вимог шляхом збору глобальних вимог від користувачів, клієнтів та інших зацікавлених сторін. Етап проектування складається з виконання набору завдань, пов'язаних із дуже конкретними агентами, після визначення цих вимог. Більш конкретно, здійснюється процес дослідження інфраструктури IoT, де об'єкти, підключені до мережі,

моделюються як зв’язані відкриті агенти (Licked Open Agents – LOA). Крім того, виконуються завдання, пов’язані з самою інфраструктурою (перехоплення повідомлень, додавання нових повідомлень, виконання робочого процесу та виявлення агентів). Етапи побудови та інтеграції представлені як етап макроскопічного рівня, на якому LOA мікроскопічного рівня інтегруються та координуються в одній мережі. Крім того, його автори розробили цей етап, щоб відповідати вимогам взаємодії LOA, співпраці, координації, обробки даних, взаємодії з користувачем та інтелектуальної поведінки. Слід зазначити, що в цій методології згадується використання інструментів, які розробник може використовувати на етапах аналізу та проектування. Однак немає доказів етапів планування, реалізації, експлуатації та обслуговування.

Робота [44] – це методологія для розробки IoT, яка відповідає V-моделі XT. У ній центром розгляду є перевірка та валідація вимог, функціональних можливостей і принципів, що керують системою. Іншими важливими етапами є визначення вимог, проектування (в деяких аспектах), інтеграція та етапи тестування. Однак вони не надають доказів щодо етапу експлуатації та технічного обслуговування.

INTER-METH [45] можна вважати ітеративною методологією, яка визначає шість послідовних етапів розробки: аналіз, проектування, впровадження, розгортання, тестування та обслуговування. Кожна ітерація може бути спрямована на вдосконалення окремих етапів, набору послідовних етапів або всього процесу, таким чином покращуючи адаптивність до нових вимог. INTER-METH – це методологія, адаптована на основі традиційної водоспадної методології, яка відрізняється тим, що вона розділяє проблему на підпроблеми для забезпечення успішного розвитку. INTER-METH, будучи заснованим на методології водоспаду, має відповідати кожній із характеристик своєї базової методології. Однак його автори не представляють інструментів, інструкцій, дій або завдань, типових для IoT, таких як розгортання апаратного забезпечення.

Автори роботи [46] представляють SERVUS, як методологію розробки IoT, спрямовану на в ирішення проблем сумісності шляхом прийняття сервіс-

орієнтованої архітектури на основі еталонної архітектури промислового Інтернету (IIRA 4.0). SERVUS займається визначенням і аналізом вимог, а також етапами аналізу та проектування. Щоб отримати вимоги та їх аналіз, вони рекомендують історії користувачів і випадки використання як основні артефакти для цього етапу. Однак немає жодних підтверджуючих доказів для етапів розробки програмного забезпечення, розгортання або експлуатації та обслуговування.

Методологія на основі MDD і MDE встановлює наступні етапи розробки:

1) аналіз бізнес-вимог,

2) визначення бізнес-логіки,

3) проектування рішення інтегрованих послуг,

4) генерація технологічного рішення,

5) модель методів трансформації.

Вони використовують рекомендації MDE та використовують такі мови, як уніфікована мова моделювання (UML) для визначення бізнес-вимог на етапі 1, і модель бізнес-процесу та нотації (BPMN) для визначення бізнес-логіки на етапі 2. Згодом на стадії 3 виходить більш уточнена модель, дотримуючись підходу SOA. Щоб отримати специфічний для платформи код на етапі 4, виконуються два кроки: по-перше, PSM виводиться з PMI, а по-друге, PSM перетворюється на код. Це одна з методологій розробки IoT, яка представляє та визначає всі основні етапи розробки такого типу системи. Однак його автори не розглядають, як розробляти інтерфейс користувача. Крім того, вони представляють набір інструментів для захоплення та аналізу FR IoT, але NFR не розглядаються.

Методологія SEM базується на метамоделях. Моделювання здійснюється з двох точок зору: з функцій, які повинна виконувати система, і з даних, з якими вона працює. Він представляє 3 етапи: етап аналізу вимог, на якому надана метамодель використовується для отримання моделі на етапі проектування, і, нарешті, етап впровадження системи. SEM – це ще одна методологія, зосереджена на створенні IoT, тому вона зосереджена на необхідних аспектах для отримання кінцевого продукту з вимогами, які вимагають користувачі.

Однак у ньому згадуються лише важливі етапи, такі як планування, визначення вимог, впровадження, а також експлуатація й обслуговування IoT. Крім того, це конкретна методологія для конкретної області.

Arrowhead представляється, як каркас. Методологія проектування, розробки та перевірки для кожної служби, системи та системи систем у структурі Arrowhead підтримує їх реалізацію, перевірку, розгортання та виконання у сумісний спосіб. Arrowhead допомагає в створенні IoT з точки зору того, що систему можна побудувати шляхом інтеграції інших систем. Хоча його автори багато згадують етапи життєвого циклу програмного забезпечення/системи, вони не надають доказів, які б скеровували розробників у виконанні їхньої діяльності.

IDeA є методологією для розробки IoT на основі системної інженерії на основі моделей (MBSE). Наданий метод базується на існуючих стандартах, до яких додано види діяльності, які вважаються найбільш відповідними для розробки додатків IoT. IDeA забезпечує високорівневі абстракції за допомогою метамоделювання як можливе рішення проблеми гетерогенності (апаратного та програмного забезпечення). Крім того, їхній метод розробки додатків IoT є мультидисциплінарним методом, у якому беруть участь усі зацікавлені сторони, залучені до процесу. Вони застосовують стандарт ISO/IEC/IEEE 15288:2015, на основі якого вони реалізують процеси життєвого циклу систем, хоча вони не визначають усі їхні фази. Хоча IDeA посилається на стандарт ISO, він не містить вказівок щодо його застосування. Крім того, немає жодних доказів, які стосуються етапу планування та, будучи методологією, заснованою на метамоделях, змушують розробників дотримуватися їх.

Методологія, запропонована у [47] базується як на функціональних (функції та служби), так і на метамоделях даних (джерела даних, атрибути та зв’язки) метамоделях і зосереджена на проектуванні IoT. Її автори розглядають визначення вимог як попередній етап і зосереджуються безпосередньо на моделюванні системи. Будучи заснованим на метамоделях, вона створює стереотипи щодо: середовища, в якому система збирається працювати, функцій,

які може запропонувати інтелектуальне середовище, яке може бути атомарним або складеним.

У метамоделях ця методологія визначає процеси, які зроблять середовище інтелектуальним. Щоб використовувати метамодель, проблема повинна бути узгоджена з метамоделлю, яка буде застосована в рішенні. Методологія чітко орієнтована на етап проектування, але немає жодних доказів того, що ця методологія спрямована на виявлення та аналіз вимог. Він також не стосується планування розвитку IoT або завершальних етапів, таких як впровадження, експлуатація та обслуговування. Крім того, не зрозумілий етап будівництва чи отримання кінцевого продукту.

Автори [48] зосередилися на ролях членів команди, щоб спробувати вирішити проблеми неоднорідності технологій, за допомогою яких можуть бути реалізовані програми IoT. Експерти в області та розробники програмного забезпечення контролюють аналіз системи та діяльність на етапі проектування. На етапі побудови та тестування системи займаються прикладні програмісти та розробники пристроїв. Нарешті, мережеві адміністратори встановлюють додаток у відповідній системі, що відповідає етапам реалізації та розгортання інших методологій. Ця методологія не стосується етапів планування, експлуатації та технічного обслуговування.

У роботі [50] пропонують інженерний підхід на основі метамоделі для систематичної розробки розумних об’єктів (SO). Етап аналізу стосується моделювання відповідних аспектів SO за допомогою метамоделі. На стадії проектування намагаються змоделювати функціональні компоненти системи, їхні зв’язки та взаємодію за допомогою моделі смарт-об’єктів, заснованої на структурі ELDA [51] і платформі ACOSO [52]. Для підтримки стадії впровадження метамодель розумних об’єктів на основі ACOSO була спеціалізована щодо платформи JADE, у результаті чого була створена метамодель JACOSO [54]. Ця метамодель висвітлює компоненти платформи JADE (людей, їх поведінку та повідомлення). Інші елементи метамоделі представляють завдання, які мають відповідно виконуватися особою,

відповідальною за конфігурацію системи, менеджером зв’язку та диспетчером пристроїв, а також визначені користувачем завдання, включені в правила виведення, які контролюють поведінка розумних об'єктів. Щоб використовувати цю метамодель, розробник повинен освоїти фреймворки та платформи, названі вище.

AMG [55] – це методологія розробки додатків IoT на основі SOA, яка складається з трьох кроків: (1) визначення абстракцій, (2) моделювання та (3) генерація коду. AMG базується на висхідному підході, оскільки він починає з конкретних моделей (конкретних послуг) для отримання абстрактних послуг. Процес абстрагування починається з описів сервісів таким чином, що спочатку отримують необхідні графічні зображення, а потім – вихідний код.

Деякі автори розглядають аналіз вимог як частину етапу аналізу системи, що є причиною, чому вони не звертаються до виявлення вимог. Однак інші автори дуже добре розділіть ці етапи та надають їм необхідного значення, оскільки системні вимоги мають бути достатньо чіткими з самого початку, щоб розробка IoT не затримувалася. Можна зробити висновок, що відсутність урахування етапу планування є широко поширеним явищем.

Інший етап, який не згадується більшістю дослідників, які розробляли методології розробки – це об’єднаний етап експлуатації та технічного обслуговування (тобто унікальний етап, що поєднує діяльність з експлуатації та технічного обслуговування).В роботі [14] представляють етап обслуговування як частину відповідних методологій. Коротко розглянемо цей етап. Можна припустити, що тут розробники розглядають цей етап завдяки підходу, на якому спирається методологія IDeA. Тобто проектування розглядається як робота, зосереджена на представленні адаптивної архітектури для IoT.

Серед методологій, які стосуються отримання програмного коду, є AMG [15] і SEM [56], зокрема в моделюванні та/або отриманні програмного забезпечення для пристроїв IoT, ігноруючи програми, які будуть служити для взаємодії між користувачем та IoT. Тут автори звертаються та згадують дизайн, орієнтований на користувача, вони не надають доказів звернення до розробки

додатків для кінцевих користувачів. Автори згадали про це, щоб отримати апаратну складову IoT і програмне забезпечення для її налаштування.

3.2 Інші пропозиції щодо розробки IoT

У таблиці 3.1 представлені основні характеристики додаткових платформ, фреймворків та інструментів для розробки IoT.

Таблиця 3.1 – Основні характеристики платформ, фреймворків та інструментів для розробки IoT

Підхід	Артефакти	Кінцевий продукт
Компоненти, шаблони, MDD	Код, конструктивна схема пристрою	Пристрій IoT: програмне забезпечення та дизайн
Компоненти, шаблони, MDD	Код, конструктивна схема пристрою	Пристрій IoT: програмне забезпечення та дизайн
Компоненти	PrIoT -core, PrIoT -API, Prior-Test, Prior-DB, PrIoT -UI	Концептуальна основа
MDA, шаблони та онтології	BPMN, CD, AD	програмне забезпечення
MDA	компоненти	Код на мовах NesC і C
Компоненти (API), MDD	Код, BD	Пристрій IoT: програмне забезпечення та дизайн

Автори [57] пропонують PrIoT, структуру розробки, яка пропонує групувати рішення сторонніх розробників для досягнення певних цілей. Фреймворк складається з трьох модулів, щоб спробувати інтегрувати різні компоненти, які складають IoT. PrIoT-core робить роботу розробників незалежною від пристроїв і деталей протоколів зв'язку. PrIoT-Lang розгортає апаратно-незалежний інтерфейс програмування. PrIoT-API фіксує найбільш практичні сценарії IoT в обмеженому вигляді. PrIoT-Test дозволяє вказати метрики для перевірки достовірності результатів, отриманих від прототипу. PrIoT-DB дозволяє нам вибрати необхідне обладнання від конкретного або загального постачальника та включити бібліотеки в проект, наприклад, щоб забезпечити обмін інформацією з іншими пристроями. Компоненти

налаштовуються через файл. PrIoT-UI представляє високорівневий графічний інтерфейс для підтримки розробників.

У роботі [58] автори пропонують структуру розгортання мобільних додатків на основі MDA. Шаблони розробки, засновані на семантичному міркуванні, надаються для цільової розробки додатків Cloud of Things (CoT) у конфігурований та адаптований спосіб. Вони також надають метамодель із бізнес-компонентами з кількома представленнями та сервісними компонентами для обміну моделями. Вони дотримуються шаблону MVC (Model-View-Controller), щоб трансформувати бізнес-моделі в сервісний компонент для налаштування хмарних служб.

COMFIT – це інтегроване середовище розробки (IDE), засноване на MDD і хмарних обчисленнях. COMFIT складається з двох модулів: (1) модуль розробки для проектування IoT з використанням адаптивних моделей високої абстракції; і (2) набір веб-додатків для керування та виконання, здатних генерувати код, спрямований на платформи Contiki та TinyOS з моделей, отриманих на стадії проектування. Варто зазначити, що COMFIT не стосується розробки додатків для кінцевого користувача.

EDG – це методологія створення інтегрованих проектів. Для цього користувачеві необхідно вказати прості вимоги до свого вбудованого програмного забезпечення. З цієї специфікації (апаратно-незалежного коду) створюється синтезована остаточна електрична схема, список необхідних матеріалів і мікропрограми, необхідні для того, щоб пристрій відповідав вимогам. Нарешті, діаграма апаратної конфігурації дозволяє нам розробити фізичний пристрій і написати апаратно-залежний код.

TinyLink – це інструмент, який слідує методології EDG. TinyLink призначений для розробників без досвіду роботи зі вбудованими системами, які можуть мати обмеження як на апаратне забезпечення, так і на користувачів. TinyLink прагне оптимізувати використання апаратного забезпечення за допомогою набору API, щоб дозволити розробникам здійснювати процес розробки «знизу вгору», на відміну від традиційних інструментів, які зазвичай

використовують підхід «зверху вниз». Крім того, TinyLink може генерувати апаратно-залежний код. Так само Autolink – це інструмент, заснований на TinyLink. На додаток до створення функціональних рішень TinyLink, Autolink звертається до NFR, таких як оцінка корисного терміну служби апаратних компонентів пристрою, часу виконання та оптимізації використання акумулятора.

Усі ці роботи підтримують розробників, коли справа доходить до програмного коду для конфігурації апаратного забезпечення IoT. Однак немає жодних доказів того, як отримати програмний код для програм, які слугуватимуть засобом взаємодії з кінцевим користувачем. Крім того, представлені інструменти потребують попередньо визначених вимог як вхідних даних для етапу проектування та подальшої генерації коду.

3.3 Архітектури для IoT

Архітектура будь-якої системи служить орієнтиром у процесі її розробки. Таким чином, важливо знати архітектури, які дослідники запропонували для розробки IoT, а також чи методологія, яку вони представляють для розробки IoT, базується на певній архітектурі, і яка є найпоширенішою. Серед найпоширеніших архітектур – багаторівнева та сервіс-орієнтована архітектури. Крім того, деякі автори представляють комбінацію обох типів, тоді як інші представляють власні архітектури, щоб надати рішення для різних типів IoT.

3.3.1 Багаторівнева архітектура

Деякі існуючі IoT були розроблені відповідно до багаторівневої архітектури, як в [59], яка складається з п’яти або шести шарів (див. рис. 3.1). Як можна помітити, між обома архітектурами є схожість. Інші подібні архітектури мають лише чотири рівні. Різниця між цими останніми архітектурами полягає в порядку їх шарів. У деяких роботах використовують чотири шари: (1) пристрій, (2) мережа, (3) проміжне програмне забезпечення та (4) додаток, тоді як в інших

його рівнями є: (1) сприйняття, (2) проміжне програмне забезпечення, (3) служби та (4) програми. Отже, нижні рівні (1) і верхні рівні (4) мають однакові цілі, тоді як другий і третій рівні міняються місцями щодо двох інших архітектур.

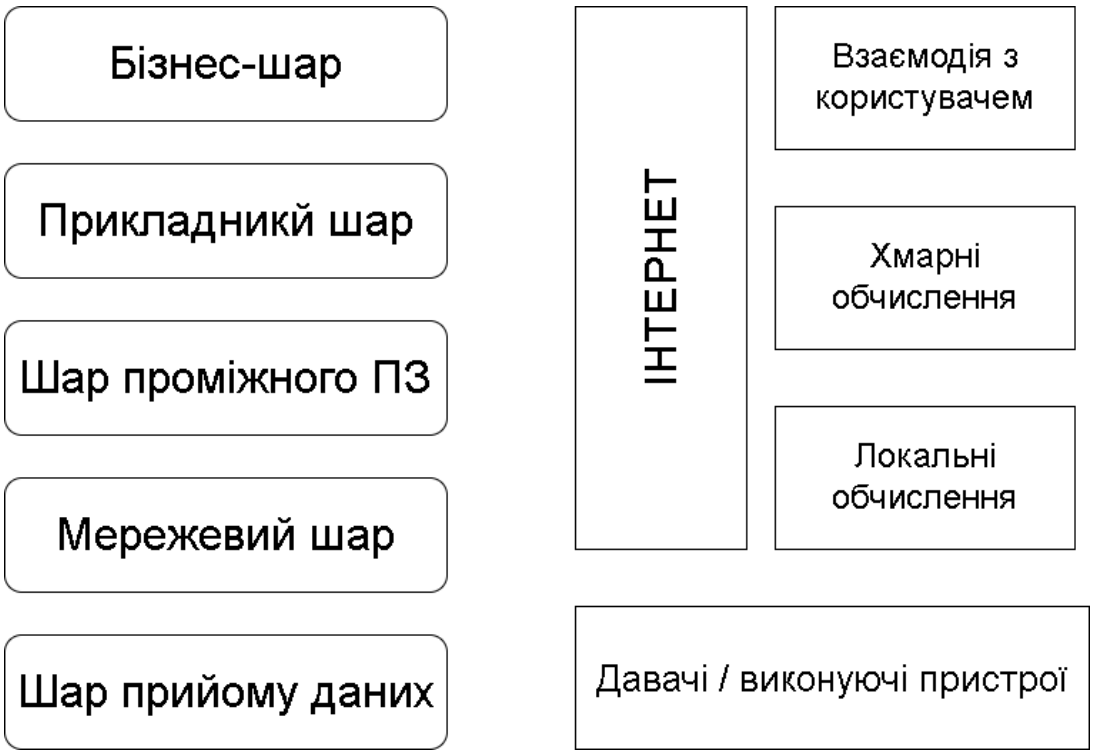

Рисунок 3.1 – Багаторівнева архітектура систем IoT

На відміну від архітектур на рисунку 3.1, набір інструментів RapIoT [60] і Mobile Health Platform [61] базуються на трирівневій архітектурі, де два рівні виконують подібні функції (керування пристроєм і додатками) в обох архітектурах, відрізняються лише на третьому рівні (комунікації). У першому випадку цей рівень виконує функції хмарної служби, яка дозволяє зберігати дані та інтегрувати їх зі сторонніми службами, тоді як у другому він складається з проміжного програмного забезпечення (сервера додатків) і веб-додаток, який з'єднує різні об'єкти фізичного рівня з іншими учасниками (медичними працівниками, лікарнями та іншими системами). Рівні RapIoT називаються вбудованим рівнем, рівнем шлюзу та серверним рівнем, а в Mobile Health Platform – рівнем фізичних об'єктів, мережевим рівнем і порталом здоров'я. Третій рівень виконує функції, дуже подібні до рівня проміжного програмного забезпечення, рівень сервісів рівень хмарної обробки.

У роботі [62] представляють IoT для управління міським трафіком, реалізований за трирівневою архітектурою клієнт/сервер. Клієнтський рівень складається з програм для кінцевих користувачів, вхідні дані яких в основному надає Pentaho BI. Бізнес - рівень складається з API, які надають дані про погоду, на додаток до рівня сервера Pentaho BI та веб-додатку. Вони використовують два менеджери баз даних: MySQL, який зберігає дані, отримані датчиками та прогнози погоди, і PostgreSQL, де дані час від часу копіюються для формування кубів OLAP для аналізу даних.

Методологія, представлена у [63], базується на IIRA v1.9. Вона визначає трирівневу схему:

1. Граничний рівень збирає дані з крайових вузлів, використовуючи безконтактну мережу. Архітектурні особливості цього рівня, включно з широтою розповсюдження, розташуванням, сферою управління та характером безконтактної мережі, змінюються залежно від конкретних випадків використання.

2. Рівень платформи отримує, обробляє та пересилає команди керування з рівня підприємства (пояснення нижче) на крайовий рівень. Він консолідує процеси та аналізує потоки даних периферійного та неграничного рівня, а також надає можливості керування пристроями та активами. Він також пропонує послуги, не пов’язані з доменом, такі як запити та аналіз даних.

3. Корпоративний рівень реалізує специфічні для домену програми, системи підтримки прийняття рішень і забезпечує інтерфейси для кінцевих користувачів, включаючи спеціалістів з експлуатації. Цей рівень отримує потоки даних від крайнього рівня та рівня платформи та видає команди керування, спрямовані на обидва боки.

У роботі [64] дизайн системи описується на доменно-залежній мові (DSL), яку вони використовують для підтримки відповідності між автоматично згенерованою моделлю BIP (поведінка, взаємодія, пріоритет) і кодом програми. BIP – це мова з формально визначеною семантикою для побудови виконуваних моделей змішаних програмно-апаратних систем (SW/HW). Модель BIP

базується на стандартах архітектури WPAN. Ця змішана архітектура складається з чотирьох рівнів абстракції, де нижній рівень визначається абстракцією апаратної архітектури, верхній рівень є абстракцією програмного забезпечення, третій рівень визначається операційною системою, а другий рівень визначається мережевим стеком і драйверами пристроїв.

3.3.2 Сервісно-орієнтовані архітектури

SOA використовується для створення програмних систем із складених, гетерогенних і автономних програмних одиниць, які називаються службами. Крім того, композиція послуг є поширеним підходом до розробки складних програмних систем. Програмні системи та програми, у свою чергу, також стають послугами. Ми називаємо це системами або програмами на основі сервісів.

У деяких роботах, розглянутих у цьому огляді, пропонують архітектуру на основі SOA, що складається з чотирьох рівнів:

1) об’єктний рівень (апаратні об’єкти, доступні в мережі);

2) мережевий рівень (інфраструктура дротового, бездротового або мобільного зв’язку);

3) сервіс (створення та керування необхідними послугами);

4) прикладний рівень (відповідальний за доставку додатків користувачам IoT).

Ця сервіс-орієнтована архітектура підтримує методології розробки з двома різними підходами: MDD та MDE.

Інша робота з цим типом змішаної архітектури визначена у [65], в основному з 3 рівнями: клієнтським або зовнішнім рівнем, комунікаційним рівнем або шлюзом API та серверним або внутрішнім рівнем. Цей останній рівень, у свою чергу, визначається мікросервісами, які надають інформацію для керування користувачами, групові концепції, пов’язані як з організаційною структурою учасників, так і з визначенням речей, і дозволяють клієнтам отримувати доступ як до значень даних, так і до графічних ресурсів.

Метод розробки IoT, запропонований у [45], базується на SOA. У першому випадку автори розглядають існування конкретних сервісів для абстрагування абстрактних сервісів для моделювання розглянутої системи. Після отримання моделі він генерує код для нової програми на основі сервісу. Цей тип архітектури (SOA) значно зменшує складність у проектуванні гетерогенної системи, такої як IoT. Іншою роботою на основі SOA є структура Arrowhead [66], де операції на різних ресурсах можуть бути згруповані в різні служби. У Arrowhead ресурсом може бути датчик температури або показання лічильника споживання енергії.

3.3.3 Інші типи архітектур

MDE4IoT [40] базується на архітектурній моделі MARTE, яка включає 3 пакети:

1) MarteFoundations, який визначає всі основні концепції, необхідні для аналізу та проектування на основі моделі систем реального часу та вбудованих систем (RT/ ESs);

2) MarteDesignModel, який охоплює від фіксації вимог до специфікації вимог, проектування та впровадження (V-цикл процесу розробки);

3) MarteAnalysisModel, який визначає конкретні абстракції моделі та відповідні анотації, які будуть використовуватися зовнішніми інструментами.

Таким чином, пакет (1) визначає загальні поняття для методів кількісного аналізу, які можна розширити для підтримки нових методів аналізу моделі RT/ES UML, тоді як пакети (2) і (3), відповідно, зосереджені на програмованості та аналізі продуктивності.

Архітектура ROOD [67] базується на MDA. Вона включає чотири рівні абстракції, упорядковані від найвищого до найнижчого рівня наступним чином: (M3) Рівень мета-метамоделі, який служить для встановлення основи для різних метамоделей; (M2) Рівень метамоделі, де DSL вказуються для визначення моделей на рівні M1; (M1) Рівень моделі, де визначаються моделі системи; і (M0) рівень екземплярів, який містить екземпляри даних для даної платформи.

Архітектура COMFIT [68] включає два модулі: (1) модуль розробки додатків (ADM) і (2) менеджер керування та виконання додатків (AMEM). ADM включає моделі PIM і PSM, а також перетворення M2M і шаблони генерації коду для використання перетворень M2T. З іншого боку, AMEM включає: (1) компонент Interface Manager, який безпосередньо підключений до ADM і відповідає за надання функціональних можливостей завантаження згенерованого коду на сервер, розміщений у хмарі; і (2) Execution Manager, який підключений як до Testbed Manager, так і до Compile Manager і розгортає служби, які випускаються через Interface Manager.

Структура, запропонована у [58] базується на архітектурі програмного забезпечення для розробки послуг мобільного зв'язку, що складається з трьох модулів:

1) інформаційний модуль для інкапсуляції пристроїв, який підтримує кілька видів бізнес-моделювання;

2) середовище викиду, що використовується для налаштування середовища програми та правил виконання бізнес-програми;

3) репозиторій ресурсів для конфігурації інформації, який призначений для з'єднання інформаційної моделі та середовища виконання.

В структурній архітектурі SDG-Pro компоненти класифікуються відповідно до їх підключення до Інтернету: компоненти Edge або Cloud. Крайовими компонентами є датчики, виконавчі механізми та комунікаційні пристрої. Комунікаційні пристрої підключаються до хмарних компонентів через програмно визначені шлюзи, які є частиною хмарних компонентів. Крім того, у складі хмарних компонентів ми знаходимо компоненти інформаційних технологій для зберігання, обробки та інтелектуального аналізу даних.

Ще одна компонентна архітектура представлена у [69]. Перший компонент – конструктор системи IoT. Другий компонент – це обчислювальні ресурси або сервер, який взаємодіє з подачею або отриманням даних від інших компонентів. Сервер виконує бізнес-логіку за запитом клієнтів Docker Engine і має інструменти моніторингу для взаємодії з компонентом супервізора. Третій

компонент складається з Docker Engine, який містить технологію обміну повідомленнями (сервер RabbitMQ). Четвертий компонент – системний супервізор IoT, який відстежує ресурси та виявляє системні збої. Цей останній компонент взаємодіє з першим, щоб оновлювати контрольовану систему відповідно до того, що вимірюють/виявляють датчики.

Методологія SEM підтримується архітектурою, заснованою на моделях, орієнтованих на побудову програмного забезпечення. Її автори представляють функціональну метамодель і метамодель даних. Інша методологія, яка підтримується цим типом архітектури, це IdeA, де розробник додатків IoT розбиває систему на функціональні компоненти, які взаємодіють один з одним, щоб відповідати системним вимогам. Моделі IDeA розкривають пристрої як служби за допомогою нотації портів. Ці сервіси будуть використовуватися інженерами додатків для створення інформаційних представлень. У цьому ж переліку ми можемо згадати пропозицію [54], яка базується на метамоделях для смарт-об’єктів як метамоделі дуже високого рівня, яка спеціально розкриває статичні та динамічні характеристики смарт-об’єктів. Метамодель на основі ELDA спеціалізується на метамоделі смарт-об’єктів високого рівня, що забезпечує функціональні компоненти системи, їхні зв’язки та взаємодії. А метамодель на основі ACOSO – це проміжне програмне забезпечення, спеціально розроблене для повного керування взаємодіючими та агентно-орієнтованими смарт-об’єктами.

У праці [70] пропонують архітектуру публікації-підписки, в якій датчики діють як видавці. Комп’ютерні послуги – це абоненти, які змушують виконавчі механізми виконувати відповідні дії (зміни в середовищі, сповіщення кінцевих користувачів і так далі) щоразу, коли надходять нові дані.

Архітектура, представлена в [71] – це мережева архітектура бездротової сенсорної мережі на основі мобільного збирача даних. Вона орієнтована на пояснення роботи та розгортання системи. Вона також забезпечує зворотний зв’язок з користувачами щодо результатів етапу аналізу вимог. З іншого боку, автори [72] представити архітектуру, яка може керувати розробкою IoT.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Аналіз небезпек при розробці програмного забезпечення

Організація робочого місця розробника ПЗ впливає на його працездатність.

У своїй діяльності розробник використовує комп’ютер, пристрої збереження інформації, а тому є необхідність забезпечення зручного доступу до всіх технічних засобів. Тому в даному розділі докладніше розглянемо відомості про систему ергономічних норм і принципів організації робочого місця, на котрому проводяться роботи зі створення модуля збору статистики.

Під робочим місцем розуміється зона, оснащена необхідними технічними засобами, у якій відбувається трудова діяльність виконавця або групи виконавців, які спільно виконують одну роботу або операцію.

Організація робочого місця полягає у виконанні заходів, які забезпечують безпечний і раціональний трудовий процес і ефективне використання знарядь та предметів праці, що підвищує продуктивність праці і знижує стомлюваність працівника.

Організація робочого місця залежить від характеру розв'язуваних задач і особливостей предметно-просторового оточення, що визначають робоче положення тіла і можливість пауз для відпочинку, типи і способи засобів відображення і керування, необхідність у засобах захисту, спецодягу, простору для налагодження і ремонту устаткування.

Одним з компонентів діяльності на робочому місці є робочі рухи. Їхня раціональна організація створює умови для зниження стомлення, резерви для підвищеної працездатності. Просторові характеристики руху оператора визначаються траєкторіями руху і розмірами моторного поля (зони досяжності).

При організації робочого місця необхідно забезпечити нормальні умови огляду. Зону огляду описує кут, вершина якого знаходиться в центрі ока, а

сторони складають границі, в яких людина при фіксованому положенні голови й ока добре розрізняє їхнє місцезнаходження.

У горизонтальній площині цей кут складає 300 – 400. При організації робочого місця кут огляду можна взяти 500 – 600, включаючи зону менш ясного огляду. Допустимий кут огляду по горизонталі 900. У вертикальній площині оптимальний кут огляду 100 вверх і 300 вниз від лінії погляду, а допустимий 300 вгору і 400 вниз від лінії погляду.

Щоб зберегти нормальну гостроту зору, робочу поверхню розташовують від очей на відстані від 0,3 м до 0,75 м. Робочі меблі повинні бути зручними для виконання робочих операцій. В даному випадку робочий стіл є основним устаткуванням. Особливо важливе значення має висота столу, його конструкція, яка повинна передбачати шухляди для розміщення інструментів, документації.

Важливе значення має конструкція робочих крісел. Погано підібрані крісла можуть бути причиною надмірної стомлюваності.

Нахил і висота крісла повинні регулюватися відповідно до висоти робочої поверхні і росту працюючого. Рекомендована ширина крісла 370 – 400 мм, глибина 370 – 420 мм, висота спинки 370 – 1000 мм від рівня крісла. Для розміщення ніг необхідно передбачити вільний простір під робочою площиною.

Праця людини, що протікає в умовах надмірного нервово-емоційного напруження, довготривалих статичних навантажень, обмеженої рухової активності призводить до неврозів, відхилень у психіці, захворювань опорно-рухового апарату, серцево-судинної системи тощо. Комп'ютери, телебачення, системи зв'язку та інші засоби, що використовують досягнення радіоелектроніки, є генераторами цілої низки електромагнітних випромінювань, вплив яких на організм людини ще не зовсім вивчений.

З широким впровадженням автоматизації та комп'ютеризації виникла потреба врахування психологічних можливостей людини, таких як швидкість реакції, особливості пам'яті та уваги, емоційний стан та ін. Поява операторської діяльності призвела до суттєвих змін у фаховій структурі праці. Зменшились фізична важкість праці, ризик виробничого травматизму, однак разом з тим, на

працюючу людину посилюється вплив нових, раніше не відомих чи мало вивчених несприятливих виробничих факторів фізичного, хімічного і особливо психофізіологічного характеру.

Проте, розвиток сучасної обчислювальної техніки відбувається не лише у бік поктащення її технічних параметрів, але також звертається увага безпеку використання цієї техніки людиною шляхом зменшення потужності випромінювачів, зменшенням рівня випромінювання з моніторів, зменшення напруг живлення, покращення ергономічних характеристик.

Таким чином, в розділі з охорони праці виконано огляд питань безпечної роботи при створенні сайту та встановлено, що умови такої роботи відповідають вимогам з охорони праці, які застосовуються в галузі інформаційних технологій.

Інформаційно-психологічні небезпеки.

Сучасні реалії постіндустріального суспільства, зумовлені значним ростом інформації, відкривають ще одну сферу життєдіяльності людини – інформаційну. Сучасні засоби комунікації і обробки інформації створили принципово нові умови існування людини, що зумовило появу грандіозного проекту об'єднання національних інформаційних і телекомунікаційних структур в глобальну інформаційну інфраструктуру.

Життєдіяльність людини реалізується одночасно зі світом природи і у специфічному для людського суспільства інформаційному середовищі, що має свої закономірності розвитку і функціонування. Інформаційна сфера стає такою ж важливою складовою суспільного життя, як економічна, виробнича, побутова, політична, військова та ін. Нові інформаційні технології, засоби масової комунікації багатократно підсилили можливості впливу на свідомість і підсвідомість як окремої людини, так і на великі групи людей та населення країни загалом.

Інформаційна сфера – сукупність таких елементів:

– об'єкти інформаційної взаємодії чи впливу;

– особисто інформація, призначена для використання суб'єктами інформаційної сфери;

– інформаційна інфраструктура, що забезпечує можливість здійснення обміну інформацією між суб'єктами;

– суспільні відносини, що складаються у зв'язку з формуванням, переданням, розповсюдженням і збереженням інформації.

Особистість, активний соціальний суб'єкт, його психіка піддаються безпосередньому впливу інформаційних чинників (передумов, що чинять опір чи утруднюють формування і функціонування адекватної інформаційно-орієнтуючої основи суспільної поведінки людини (життєдіяльності у суспільстві)), які трансформуються, через його поведінку, діяльність (бездіяльність), здійснюють деструктивний, дисфункційний вплив на його життєдіяльність.

До основних загроз інформаційно-психологічної безпеки відносять можливість настання негативних наслідків для суб'єктів, що піддаються інформаційно-психологічному впливу, які виражаються в таких формах:

– нанесення шкоди здоров'ю людини;

– блокування на неусвідомленому рівні волі, волевиявлення людини, штучне привиття їй синдрому залежності;

– втрата здатності до політичної, культурної, моральної самоіндентифікації людини;

– маніпуляція суспільною свідомістю;

– руйнування єдиного інформаційного і духовного простору України, традиційних устроїв суспільства і суспільної моральності, а також порушення інших життєво важливих інтересів особистості, суспільства, держави.

Наприклад, культ жорстокості, насильства, порнографії, розбещеності тощо, які пропагують у засобах масової інформації, друкованих виданнях, комп'ютерних іграх, мережі Інтернет веде до неусвідомленого бажання у підлітків і молоді, а також дорослих з нестійкою психікою, копіювати запропоновані моделі поведінки. Цей вид пропаганди знижує рівень порогових обмежень і правових заборон, що поряд з іншими умовами відкриває шлях для

багатьох правопорушень. Це своєю чергою наносить непоправну шкоду не тільки окремій особистості, але й суттєві збитки національним інтересам країни.

Отже, джерелом інформаційно-психологічної небезпеки є та частина інформаційного середовища, яка через визначені причини неадекватно відображає реалії, вводить в оману людину, засліплює її ілюзією.

Інформаційно-психологічні загрози зумовлені розробкою, виготовленням, розповсюдженням та використанням суб'єктами негативних інформаційно-психологічних впливів, спеціальних засобів і методів такого впливу.

4.2 Концепція інформаційно-психологічної безпеки.

Сучасне розуміння безпеки в контексті врахування відношення інтересів особистості, суспільства і держави висуває завдання розгляду нового аспекту цієї проблеми – безпеки в інформаційній сфері життєдіяльності людини, тобто інформаційно-психологічної безпеки.

В інформаційному середовищі, що є складовим системним утворенням, виділяється процесуальна складова як найбільш динамічна і змінна її частина – інформаційно-комунікативні процеси, які активно впливають на індивідуальну, групову і суспільну психологію (індивідуальну, групову, масову свідомість). Маніпулюючи станом інформаційного середовища, змінюється стан духовної сфери суспільства, деформація і деструктивні зміни якої у формі психоемоційної і соціальної напруженості, спотворених норм і неадекватних соціальних стереотипів і установок, оманливих і неприродних орієнтацій та цінностей. Це своєю чергою впливає на стан і процеси у всіх основних сферах суспільного життя, в тому числі політичній і економічній.

Вперше у пострадянському просторі про проблему інформаційно-психологічної безпеки було зазначено в листопаді 1995 р. на науково-практичній конференції, організованій Інститутом психології Російської академії наук. На цій та подальших конференціях було розкрито роль знання технологій інформаційно-психологічного впливу, метою якого є маніпуляція, для

вироблення напрямів реформування психологічного захисту особистості і особистої інформаційно-психологічної безпеки.

Інформаційно-психологічну безпеку особистості визначають такими основними причинами.

Зростання тиску інформаційного середовища визначає необхідність формування нових механізмів та засобів виживання людини як особистості й активного соціального суб'єкта у сучасному суспільстві.

Взаємодія психіки людини з інформаційним середовищем відрізняється якісною специфікою і не має аналогів у комунікації інших біологічних, технічних, соціальних і соціотехнічних структур.

Основною і центральною "мішенню" інформаційного впливу є людина, її психіка.

Отже, інформаційно-психологічну безпеку можливо розглядати як стан захищеності особистості, різних соціальних груп і об'єднань людей від дій, впливів, які здатні проти їхньої волі і бажання змінити психічні стани та психологічні характеристики людини, модифікувати її поведінку і обмежувати свободу вибору, зумовило потребу переосмислення інформаційної взаємодії, а також деяких інших соціально-психологічних процесів і явищ у сучасному суспільстві.

Інформаційно-психологічна безпека – стан захищеності окремих осіб чи груп осіб від негативних інформаційно-психологічних впливів і пов'язаних з цим інших життєво важливих інтересів особистості, суспільства, держави в інформаційному середовищі.

Негативний інформаційно-психологічний вплив – процес зміни психічних станів і характеристик людей під впливом інформаційно-комунікативних процесів як динамічного компонента інформаційного середовища. Цей вплив спрямований на людину чи групу осіб (у тому числі без їхньої згоди) з метою примусу до визначеної поведінки, оцінки ситуації, керування та корекції індивідуальної та колективної свідомості. Він здійснюється з використанням

спеціальних засобів і методів впливу на психіку людини, унаслідок чого він приводить до негативних наслідків для особистості, суспільства і держави.

Спеціальні засоби впливу – технічні і програмні засоби, що використовують для використання з метою негативного інформаційно-психологічного впливу на людину чи групу людей.

Спеціальні методи впливу – послідовність прийомів впливу на психіку людини, використання яких приводить до негативних наслідків для особистості, суспільства та держави.

Головним об'єктом забезпечення інформаційно-психологічної безпеки в інформаційному середовищі у сфері індивідуальної безпеки є усвідомлення інформації, здатність людини адекватно сприймати навколишню дійсність, своє місце в зовнішньому світі, формувати відповідно до свого життєвого досвіду визначені переконання і приймати стосовно них рішення.

Інформаційно-психологічна безпека має спиратися на стандарти інформаційно-психологічної безпеки – затверджені у визначеному порядку інформаційно-психологічного впливу, який не викликає негативних наслідків для психіки людини.

ВИСНОВКИ

У цій роботі обговорювалися важливі аспекти та кроки, які стосуються існуючі методології або які повинна враховувати методологія для розробки IoT, а також настанови щодо їх впровадження. Важливим аспектом, який відрізняє IoT від інших інформаційних систем, є те, що вони складаються з об'єктів (речей), які автономно взаємодіють один з одним, розглядаючи людей як інші об'єкти або елементи системи. Для цього IoT повинні покладатися на мережі датчиків/приводів і ефективний бездротовий зв'язок. Отже, для розробки IoT важливо проаналізувати існуючі та доступні технології та навіть розробити необхідні пристрої, якщо це можливо, щоб допомогти відповідати не лише функціональні вимоги, але і нефункціональні.

Підходи на основі моделі, тобто MDE, MDD і MDA, є одними з найбільш широко використовуваних методологій для розробки IoT. Однак дуже мало з цих методологій мають справу з виявленням та аналізом вимог, і, за винятком однієї з них, жодна інша не стосується фази обслуговування та фази вилучення. Інші аспекти, які більшість розглянутих методологій не розглядають або не згадують, пов'язані з NFR. Успіх застосування цих підходів може бути пов'язаний з тим, що вони допомагають вирішити проблему неоднорідності технології, пов'язану з розробкою IoT.

Ми вважаємо, що процес розробки програмного забезпечення повинен охоплювати кожен із етапів життєвого циклу системи, від виявлення та аналізу вимог до її демонтажу. Однак у цьому типі системи, як і в інших у деяких випадках, клієнт не має зрозумілих вимог на початку, тому вважається, що розробники повинні представити ранні прототипи. Для цього повинні бути присутні підходи на основі моделі для швидкого створення програмного забезпечення, а отже, також і RP. Крім того, RP і Agile методології найбільш широко використовуються в розробці IoT. Таким чином, дослідники повинні розробити методологію для розробки IoT, яка збирає найкращі практики модельних підходів (наприклад, MDD, MDE та MDA), PR, а також 4 цінності та

12 принципів гнучкого маніфесту, на якому гнучкі методології (такі як Scrum і XP, серед інших).

Жодна з методологій розробки IoT, запропонованих у літературі та представлених у цьому документі, не використовувалася розробниками чи дослідниками, окрім їхніх власних авторів. Більшість документів, які представляють розвиток деяких IoT, використовують Scrum як єдину методологію, а деякі з них поєднують її з іншими, такими як XP, RP і Kanban.

Фреймворки розробки, які використовуються для розробки IoT, здебільшого стосуються моделювання та генерації коду для пристроїв IoT, не звертаючись до створення програм, які пропонують інтерфейс із кінцевим користувачем. Ці інфраструктури розробки також повинні підтримувати розробку таких програм. Важливим досягненням у цій галузі було б створити інструмент, який полегшує роботу розробників IoT таким чином, щоб він став універсальним для всіх них. Щоб досягти цієї мети, згаданий інструмент має враховувати функції деяких інструментів, представлених у цьому огляді, і додати інші, яких наразі немає в жодного з них.

Найпоширенішим архітектурним стилем у всіх розглянутих роботах є шаровий. Зазвичай обробка даних відбувається від нижнього рівня, відомого як фізичний рівень, рівень сприйняття або датчиків/виконавчих механізмів (серед інших імен), до найвищого, відомого як програма, взаємодія з користувачем, користувач або хмарна обробка (серед інших імен), а взаємозв’язок має бути присутнім між усіма його шарами. Архітектура служить керівництвом для підтримки роботи розробників для виконання FR і NFR системи, а отже, для отримання якісного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fortino, G.; Savaglio, C.; Spezzano, G.; Zhou, M. Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. IEEE Trans. Syst. Man Cybern. Syst. 2021, 51, 223–236.

2. Bouanaka, C.; Benlahrache, N.; Benhamaid, S.; Bouhamed, E. A Review of IoT Systems Engineering: Application to the Smart Traffic Lights System. In Proceedings of the 4th International Conference on Advanced Aspects of Software Engineering, ICAASE 2020, Constantine, Algeria, 28–30 November 2020; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2020.

3. Bodnarchuk, I., Lisovyi, V., Kharchenko, O., & Galai, I. (2018, September). Adaptive method for assessment and selection of software architecture in flexible techniques of design. In 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 292-297). IEEE.

4. Bodnarchuk, I., Duda, O., Kharchenko, A., Kunanets, N., Matsiuk, O., & Pasichnyk, V. (2020). Choice Method of Analytical Platform for Smart City (No. 4374). EasyChair.

5. Kharchenko, A., Raichev, I., Bodnarchuk, I., & Matsiuk, O. (2021, October). The Survey of Global Software Design Processes. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T) (pp. 291-294). IEEE.

6. Fowler, M., & Highsmith, J. (2001). The agile manifesto. Software development, 9(8), 28-35.

7. Hijazi, H., Khdour, T., & Alarabeyyat, A. (2012). A review of risk management in different software development methodologies. International Journal of Computer Applications, 45(7), 8-12.

8. Jones, T. S., & Richey, R. C. (2000). Rapid prototyping methodology in action: A developmental study. Educational Technology Research and Development, 48(2), 63-80.

9. Iqbal, J., Omar, M., & Yasin, A. (2019). The impact of agile methodologies and cost management success factors: An empirical study. Baghdad Science Journal, 16(2), 496-504.

10. Soares, D.; da Silva, F.J.; Ramos, S.C.F.; Kirytopoulos, K.; Sá, J.C.; Ferreira, L.P. Identifying Barriers in the Implementation of Agile Methodologies in Automotive Industry. Sustainability 2022, 14, 5453.

11. Gea, T., Paradells, J., Lamarca, M., & Roldan, D. (2013, July). Smart cities as an application of internet of things: Experiences and lessons learnt in barcelona. In 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (pp. 552-557). IEEE.

12. Yelamarthi, K., Aman, M. S., & Abdelgawad, A. (2017). An Application-Driven Modular IoT Architecture. Wireless Communications and Mobile Computing, 2017(1), 1350929.

13. Nugra, H.; Abad, A.; Fuertes, W.; Galarraga, F.; Aules, H.; Villacis, C.; Toulkeridis, T. A Low-Cost IoT Application for the Urban Traffic of Vehicles, Based on Wireless Sensors Using GSM Technology. In Proceedings of the IEEE International Symposium on Distributed.

14. Guerra Terán, P.; Plua, R.K. Home Automation Application for the Monitoring and Control of an Electric Water Heater Using AWS Technology. In Proceedings of the IEEE 38th Central America and Panama Convention, CONCAPAN 2018, San Salvador, El Salvador, 7–9 November 2018; IEEE: San Salvador, El Salvador, 2018; pp. 1–6.

15. Fortino, G., Savaglio, C., Palau, C. E., De Puga, J. S., Ganzha, M., Paprzycki, M., ... & Llop, M. (2018). Towards multi-layer interoperability of heterogeneous IoT platforms: The INTER-IoT approach. Integration, interconnection, and interoperability of IoT systems, 199-232.

16. Guerrero-Ulloa, G., Hornos, M. J., & Rodríguez-Domínguez, C. (2019, December). TDDM4IoTS: a test-driven development methodology for Internet of Things (IoT)-based systems. In International Conference on Applied Technologies (pp. 41-55). Cham: Springer International Publishing.

17. Bouanaka, C., Benlahrache, N., Benhamaid, S., & Bouhamed, E. (2020, November). A Review of IoT Systems Engineering: Application to the Smart traffic lights system. In 2020 International Conference on Advanced Aspects of Software Engineering (ICAASE) (pp. 1-8). IEEE.

18. Fortino, G.; Savaglio, C.; Spezzano, G.; Zhou, M. Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. IEEE Trans. Syst. Man Cybern. Syst. 2021, 51, 223–236.

19. Barker, T.T. Documentation for Software and IS Development. In Encyclopedia of Information Systems; Academic Press: Cambridge, MA, USA, 2003; pp. 683–693.

20. Yourdon, E. Modern structured analysis, 1989. Englewood Cliffs, 78-94.

21. Odell, J. J. (1998). Advanced object-oriented analysis and design using UML (Vol. 12). Cambridge University Press.

22. Saleh, S. M., Huq, S. M., & Rahman, M. A. (2019, February). Comparative study within Scrum, Kanban, XP focused on their practices. In 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE) (pp. 1-6). IEEE.

23. Pico-Valencia, P., Holgado-Terriza, J. A., & Paderewski, P. (2019). A systematic method for building internet of agents applications based on the linked open data approach. Future generation computer systems, 94, 250-271.

24. Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. IEEE software, 17(4), 26-32.

25. Project Management Institute. (2021, July). A guide to the project management body of knowledge (PMBOK® guide)–Seventh edition and the standard for project management. Project Management Institute.

26. Albadarneh, A., Albadarneh, I., & Qusef, A. (2015, November). Risk management in Agile software development: A comparative study. In 2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT) (pp. 1-6). IEEE.

27. Beck, K. (2022). Test driven development: By example. Addison-Wesley Professional.

28. Yatsyshyn, V., Pastukh, O., Lutskiv, A., Tsymbalistyy, V., & Martsenko, N. (2022). A Risks management method based on the quality requirements communication method in agile approaches. In ITTAP (pp. 1-10).

29. Яцишин, В. В., Петрик, Н. М., & Марковець, О. О. (2017). Управління вимогами до програмного забезпечення на основі моделі Requirements management maturity. Збірник тез доповідей VI Міжнародної науково-технічної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “, 2, 146-147.

30. Alaba, F. A., Othman, M., Hashem, I. A. T., & Alotaibi, F. (2017). Internet of Things security: A survey. Journal of Network and Computer Applications, 88, 10-28.

31. Kettunen, P., & Laanti, M. (2017). Future software organizations–agile goals and roles. European Journal of Futures Research, 5(1), 16.

32. Ozkan, N., Bal, S., Erdogan, T. G., & Gök, M. Ş. (2022, September). Scrum, Kanban or a mix of both? A systematic literature review. In 2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS) (pp. 883-893). IEEE.

33. dos Santos, M. V. M., da Silva, P. D. B., Otero, A. G. L., Wisnieski, R. T., Goncalves, G. S., Maria, R. E., ... & da Cunha, A. M. (2016). Applying scrum in an interdisciplinary project for fraud detection in credit card transactions. In Information Technology: New Generations: 13th International Conference on Information Technology (pp. 461-471). Springer International Publishing.

34. Schmidt, D. C. (2006). Model-driven engineering. Computer-IEEE Computer Society-, 39(2), 25.

35. Cabot, J. Clarifying Concepts: MBE vs MDE vs MDD vs MDA. Available online: https://modeling-languages.com/clarifyingconcepts-mbe-vs-mde-vs-mdd-vs-mda/ (accessed on 20 October 2024).

36. Schmidt, D. C. (2006). Model-driven engineering. Computer-IEEE Computer Society-, 39(2), 25.

37. Lekidis, A.; Stachtiari, E.; Katsaros, P.; Bozga, M.; Georgiadis, C.K. Model-Based Design of IoT Systems with the BIP Component Framework. Softw. Pract. Exp. 2018, 48, 1167–1194.

38. Brambilla, M.; Umuhoza, E.; Acerbis, R. Model-Driven Development of User Interfaces for IoT Systems Via Domain-Specific Components and Patterns. J. Internet Serv. Appl. 2017, 8, 14.

39. Harbouche, A.; Djedi, N.; Erradi, M.; Ben-Othman, J.; Kobbane, A. Model Driven Flexible Design of a Wireless Body Sensor Network for Health Monitoring. Comput. Netw. 2017, 129, 548–571.

40. Fortino, G.; Russo, W. ELDAMeth: An Agent-Oriented Methodology for Simulation-Based Prototyping of Distributed Agent Systems. Inf. Softw. Technol. 2012, 54, 608–624.

41. Wang, Z.; Cui, L.; Guo, W.; Zhao, L.; Yuan, X.; Gu, X.; Tang, W.; Bu, L.; Huang, W. A Design Method for an Intelligent Manufacturing and Service System for Rehabilitation Assistive Devices and Special Groups. Adv. Eng. Inform. 2022, 51, 101504.

42. Schauer, P.; Falas, Ł. Adaptation-Enabled Architecture for Internet of Things Systems. Lect. Notes Netw. Syst. 2021, 182, 195–204.

43. Pico-Valencia, P.; Holgado-Terriza, J.A.; Paderewski, P. A Systematic Method for Building Internet of Agents Applications Based on the Linked Open Data Approach. Future Gener. Comput. Syst. 2019, 94, 250–271.

44. Gogineni, S.K.; Riedelsheimer, T.; Stark, R. Systematic Product Development Methodology for Customizable IoT Devices. In Proceedings of the Procedia CIRP, Póvoa de Varzim, Portgal, 8–10 May 2019; Elsevier B.V.: Amsterdam, The Netherlands, 2019; Volume 84, pp. 393–399.

45. Fortino, G.; Savaglio, C.; Palau, C.E.; de Puga, J.S.; Ghanza, M.; Paprzycki, M.; Montesinos, M.; Liotta, A.; Llop, M. Towards Multi-Layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach. In

Integration, Interconnection, and Interoperability of IoT Systems; Springer: Cham, Switzerland, 2018; pp. 199–232.

46. Usländer, T., & Batz, T. (2018). Agile service engineering in the industrial Internet of Things. Future Internet, 10(10), 100.

47. Cicirelli, F.; Fortino, G.; Guerrieri, A.; Spezzano, G.; Vinci, A. Metamodeling of Smart Environments: From Design to Implementation. Adv. Eng. Inform. 2017, 33, 274–284.

48. Patel, P.; Cassou, D. Enabling High-Level Application Development for the Internet of Things. J. Syst. Softw. 2015, 103, 62–84.

49. Fortino, G.; Guerrieri, A.; Russo, W.; Savaglio, C. Integration of Agent-Based and Cloud Computing for the Smart Objects-

50. Oriented IoT. In Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hsinchu, Taiwan, 21–23 May 2014; pp. 493–498.

51. Fortino, G.; Garro, A.; Mascillaro, S.; Russo, W. Using Event-Driven Lightweight DSC-Based Agents for MAS Modelling. Int. J. Agent-Oriented Softw. Eng. 2010, 4, 113–140.

52. Fortino, G.; Guerrieri, A.; Russo, W.; Savaglio, C. Integration of Agent-Based and Cloud Computing for the Smart Objects-

53. Oriented IoT. In Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hsinchu, Taiwan, 21–23 May 2014; pp. 493–498.

54. Fortino, G. Agents Meet the IoT: Toward Ecosystems of Networked Smart Objects. IEEE Syst. Man Cybern. Mag. 2016, 2, 43–47.

55. Sulistyo, S. Software Development Methods in the Internet of Things. In Information and Communication Technology. ICT-EurAsia 2013. Lecture Notes in Computer Science; Mustofa, K., Neuhold, E.J., Tjoa, A.M., Weippl, E.Y.I., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7804, pp. 50–59.

56. Cicirelli, F.; Fortino, G.; Guerrieri, A.; Spezzano, G.; Vinci, A. Metamodeling of Smart Environments: From Design to Implementation. Adv. Eng. Inform. 2017, 33, 274–284.

57. Pawar, N.; Bourgeau, T.; Chaouchi, H. PrIoT: Prototyping the Internet of Things. In Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 6–8 August 2018; IEEE: Barcelona, Spain, 2018; pp. 216–223.

58. Cai, H.; Gu, Y.; Vasilakos, A.V.; Xu, B.; Zhou, J. Model-Driven Development Patterns for Mobile Services in Cloud of Things. IEEE Trans. Cloud Comput. 2018, 6, 771–784.

59. Vashi, S.; Ram, J.; Modi, J.; Verma, S.; Prakash, C. Internet of Things (IoT): A Vision, Architectural Elements, and Security Issues. In Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017, Nadu, India, 10–11 February 2017; IEEE: Palladam, India, 2017; pp. 492–496.

60. Mora, S.; Gianni, F.; Divitini, M. RapIoT Toolkit: Rapid Prototyping of Collaborative Internet of Things Applications. In Proceedings of the International Conference on Collaboration Technologies and Systems, CTS 2016, Orlando, FL, USA, 31 October–4 November 2016; IEEE: Orlando, FL, USA, 2017; pp. 438–445.

61. Al-Taee, M.A.; Al-Nuaimy, W.; Al-Ataby, A.; Muhsin, Z.J.; Abood, S.N. Mobile Health Platform for Diabetes Management Based on the Internet-of-Things. In Proceedings of the Jordan Conference on Applied Electrical Engineering and Computing Technologies, AEECT 2015, Amman, Jordan, 3–5 November 2015; IEEE: Amman, Jordan, 2015; pp. 1–5.

62. Fuertes, W.; Carrera, D.; Villacis, C.; Toulkeridis, T.; Galarraga, F.; Torres, E.; Aules, H. Distributed System as Internet of Things for a New Low-Cost, Air Pollution Wireless Monitoring on Real Time. In Proceedings of the 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2015, Chengdu, China, 14–16 October 2015; pp. 58–67.

63. Usländer, T.; Batz, T. Agile Service Engineering in the Industrial Internet of Things. Future Internet 2018, 10, 100.

64. Lekidis, A.; Stachtiari, E.; Katsaros, P.; Bozga, M.; Georgiadis, C.K. Model-Based Design of IoT Systems with the BIP Component Framework. Softw. Pract. Exp. 2018, 48, 1167–1194.

65. Brambilla, M.; Umuhoza, E.; Acerbis, R. Model-Driven Development of User Interfaces for IoT Systems Via Domain-Specific Components and Patterns. J. Internet Serv. Appl. 2017, 8, 14.

66. Varga, P.; Blomstedt, F.; Ferreira, L.L.; Eliasson, J.; Johansson, M.; Delsing, J.; Martínez de Soria, I. Making System of Systems Interoperable – The Core Components of the Arrowhead Framework. J. Netw. Comput. Appl. 2017, 81, 85–95.

67. Corredor, I.; Bernardos, A.M.; Iglesias, J.; Casar, J.R. Model-Driven Methodology for Rapid Deployment of Smart Spaces Based on Resource-Oriented Architectures. Sensors 2012, 12, 9286–9335.

68. de Farias, C.M.; Brito, I.C.; Pirmez, L.; Delicato, F.C.; Pires, P.F.; Rodrigues, T.C.; dos Santos, I.L.; Carmo, L.F.R.C.; Batista, T. COMFIT: A Development Environment for the Internet of Things. Future Gener. Comput. Syst. 2017, 75, 128–144.

69. Schauer, P.; Falas, Ł. Adaptation-Enabled Architecture for Internet of Things Systems. Lect. Notes Netw. Syst. 2021, 182, 195–204.

70. Chauhan, S.; Patel, P.; Delicato, F.C.; Chaudhary, S. A Development Framework for Programming Cyber-Physical Systems. In Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS 2016, Austin, TX, USA, 16 May 2016; Association for Computing Machinery, Inc.: New York, NY, USA, 2016; pp. 47–53.

71. Harbouche, A.; Djedi, N.; Erradi, M.; Ben-Othman, J.; Kobbane, A. Model Driven Flexible Design of a Wireless Body Sensor Network for Health Monitoring. Comput. Netw. 2017, 129, 548–571.

72. Wang, Z.; Cui, L.; Guo, W.; Zhao, L.; Yuan, X.; Gu, X.; Tang, W.; Bu, L.; Huang, W. A Design Method for an Intelligent Manufacturing and Service System for Rehabilitation Assistive Devices and Special Groups. Adv. Eng. Inform. 2022, 51, 101504.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року

УДК 004.42; 004.9; 004.056.5

Тетяна Щеснюк
Тернопільський національний технічний університет імені Івана Пулюя

ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ГНУЧКИХ ТЕХНОЛОГІЙ РОЗРОБКИ ДЛЯ IoT-СИСТЕМ

Tetiana Shchesniuk

PARTICULARITIES OF AGILE DEVELOPMENT TECHNOLOGIES USING FOR IoT SYSTEMS

Застосування технологій гнучкої розробки до систем на основі Інтернету речей (Internet of Things – IoT) передбачає врахування ряду унікальних аспектів через складність і різноманітність IoT-систем. Гнучкі принципи, такі як ітеративна розробка, співпраця з клієнтами та адаптивність, добре узгоджуються з динамічною природою проектів IoT. Однак інтеграція програмного забезпечення з апаратним та з мережевими компонентами створює певні проблеми [1].

Одним з критичних аспектів є залежність від апаратного забезпечення. Системи IoT містять різноманітні датчики та виконавчі механізми, які мають довший цикл розробки, ніж програмне забезпечення. Гнучкі методології необхідно адаптувати для врахування цих розширених часових рамок, часто вимагаючи паралельних робочих процесів, де розробка обладнання та програмного забезпечення розвивається незалежно, але залишається тісно скоординованою.

Ще одним викликом є міждисциплінарна співпраця. Проекти IoT часто включають різні команди, включаючи розробників програмного забезпечення, інженерів апаратного забезпечення та дослідників даних. Гнучкі структури повинні сприяти ефективній комунікації та інтеграції між цими командами, часто вимагаючи спеціальних інструментів і практик для управління цими взаємозалежностями.

Багато систем IoT вимагають обробки в реальному часі та швидкості реагування, що необхідно враховувати під час планування спринту та тестування. Ітеративна розробка повинна гарантувати, що кожен крок відповідає стандартам продуктивності та надійності, важливих для додатків IoT. Тестування та розгортання в проектах IoT також істотно відрізняються. Тестування часто вимагає інтеграції з фізичними пристроями, змодельованим середовищем або периферійними обчислювальними установками, що може ускладнити конвеєри безперервної інтеграції та доставки (CI/CD). Розгортання, особливо для пристроїв у польових умовах, включає оновлення по повітрю (over-the-air – OTA), які мають бути надійними та безпечними, щоб уникнути системних збоїв.

Безпека та конфіденційність даних викликають додаткові проблеми. Гнучка розробка для IoT повинна завчасно і постійно інтегрувати методи безпеки, усуваючи вразливості, які можуть виникати через взаємопов'язані пристрої та мережі.

Таким чином, незважаючи на те, що гнучка розробка забезпечує міцну основу для проектів Інтернету речей, її застосування потребує індивідуальних практик для усунення залежностей апаратного забезпечення, міждисциплінарної співпраці, вимог до роботи системи у режимі реального часу та викликів безпеки.

Література

1. 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.