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

бакалавр
(назва освітнього ступеня)

на тему: Розробка методу класифікації типів артефактів для покращення управління програмних проєктів.

Виконав(ла): студент(ка) 4 курсу, групи СТ-41
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

(підпис) Беш Т.Р. (прізвище та ініціали)

Керівник (підпис) Никитюк В.В. (прізвище та ініціали)

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

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

Рецензент (підпис) Стоянов Ю.М. (прізвище та ініціали)

Тернопіль
2023

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)
«22» червня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

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

студенту Беш Тарас Романович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розробка методу класифікації типів артефактів для покращення управління програмних проєктів**

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

Затверджені наказом ректора від «*07*» *лютого* 2023 року № *4/7-134*

2. Термін подання студентом завершеної роботи 22 червня 2023 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ 1.1 Роль гнучких методологій в сучасній індустрії розробки програмних продуктів 1.2 Характеристики гнучких методологій розробки програмного продукту РОЗДІЛ 2. КЛАСИФІКАЦІЯ ТИПІВ АРТЕФАКТІВ У ГНУЧКИХ ПРОЕКТАХ 2.1 Дослідний набір даних 2.2 Результати опрацювання даних 2.3 Перевірка моделі РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ 3.1 Поняття та об'єкт аналізу технічної безпеки 3.2 Розрахунок захисного заземлення ВИСНОВОК ПЕРЕЛІК ПОСИЛАНЬ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., к.т.н., доц.		

7. Дата видачі завдання 24 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.23-27.01.23	*Виконано*
2.	Підбір джерел по темі роботи	28.01.23 – 01.04.23	*Виконано*
3.	Оформлення першого розділу	15.04.2023	*Виконано*
4.	Оформлення другого розділу	30.04.2023	*Виконано*
5.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи охорони праці»	15.05.2023	*Виконано*
6.	Оформлення кваліфікаційної роботи	07.06.2023	*Виконано*
7.	Перевірка на плагіат	07.06.2023	*Виконано*
8.	Нормоконтроль	09.06.2023	*Виконано*
9.	Попередній захист кваліфікаційної роботи	11.06.2023	*Виконано*
10.	Захист кваліфікаційної роботи	22.06.2023	

Студент _____ Беш Т.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Никитюк В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка методу класифікації типів артефактів для покращення управління програмних проєктів // Кваліфікаційна робота освітнього рівня "Бакалавр" // Беш Тарас Романович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СТ-41 // Тернопіль, 2023 // с. – 59, рис. – 12, табл. – 4, кресл. – 12, бібліогр. – 21.

Ключові слова: програмне забезпечення; управління проєктами; збір інформації; встановлення правил; набуття знань; інженерія програмного забезпечення.

Управління процесами, пов’язаними з розробкою програмного забезпечення, залишається серйозною проблемою, але це також дає можливість запровадити вдосконалення, які можуть зменшити потребу в ресурсах, необхідних для успішного завершення проектів. У роботі представлено нову концепцію для класифікації різних типів проектних завдань з метою більш ефективного використання зібраних даних у системах підтримки управління в IT-галузі.

Досліджуються методи гнучкого управління, які зараз використовуються, визнаючи неминучість змін під час виконання проекту. На основі цих методів, робота пропонує набори завдань, специфічні для розробки програмного забезпечення.

Завдяки використанню статистичних методів для створення завдань і агрегування результатів ітеративно та поступово, аналіз стає більш точним, забезпечуючи краще планування роботи над проектом, оптимальний склад команди та виявлення вузьких місць, які можуть прискорити успішне завершення проекту. Класифікація типів завдань і відповідні значення, необхідні

для планування подальшої роботи, виводяться з даних реального світу, отриманих від проектів програмного забезпечення в IT-індустрії.

ANNOTATION

Development of Artifacts Types Classification Method of to Improve Software Projects Management // Qualification work of the educational level "Bachelor" // Taras Besh // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group CT-41 // Ternopil, 2023 // p. – 59 , fig. – 12, tables – 4, references – 21, posters – 12.

Keywords: software; project management; information gathering; rules mining; acquisition of knowledge; software engineering.

Managing software development processes remains a major challenge, but it also provides an opportunity to introduce improvements that can reduce the resource requirements needed to successfully complete projects. The paper presents a new concept for classifying different types of project tasks with the aim of more effective use of collected data in management support systems in the IT industry.

Agile management techniques currently in use are explored, recognizing the inevitability of change during project execution. Based on these methods, the work offers sets of tasks specific to software development.

Using statistical methods to create tasks and aggregate results iteratively and incrementally, the analysis becomes more accurate, providing better planning of work on the project, optimal composition of the team and identification of bottlenecks that can accelerate the successful completion of the project. The classification of task types and the corresponding values necessary for planning the next work are derived from real-world data obtained from software projects in the IT industry.

ЗМІСТ

ВСТУП

Сучасна інтенсивність змін навколишньої дійсності внаслідок технічного прогресу робить управління економічними одиницями надзвичайно вимогливим і складним. Тому необхідно швидко й ефективно реагувати на зміни, які створюють нові умови для підприємницької діяльності. В епоху інженерії знань інформаційні системи, спільно створені і пов'язані з ними інформаційні технології є надзвичайно цінними і нерозривно пов’язані зі знаннями [1].

Корисність ІТ-систем величезна і безпосередньо впливає на підвищення можливостей менеджменту підрозділами компаній шляхом використання їх інноваційності та технологічного потенціалу, що має велике значення при прийнятті стратегічних бізнес-рішень. Глобалізація, ера електронної економіки, комп’ютеризація – актуальні теми сучасності господарської діяльності. Тому виробництво ІТ-продукту є настільки фундаментальним.

Управління виробництвом ІТ-продуктів є новим науковим і технологічним напрямком, що виник на стику між інформатикою та інформаційним менеджментом. Зацікавленими є всі суб’єкти господарювання, які будують свою діяльність на ІТ-рішеннях, в практичних результатах досліджень з даної дисципліни.

Крім того, слід зазначити, що опосередковано від усіх видів удосконалень у розробці та управлінні програмним забезпеченням діяльність, яка використовує будь-яке ІТ-програмне забезпечення, принесе користь. Результатом цифровізації економіки є потреба в надійності та безпеці нових програм, ІТ-систем або послуг. Недостатня надійність програмного забезпечення може спричинити величезні втрати. Тому так важливо розробляти та створювати програмне забезпечення, яке є максимально надійним.

Ефективність роботи команд, які виробляють програмне забезпечення на основі циклічних інкрементних моделей життєвого циклу, є настільки важливою задачею.

Процес розробки програмного забезпечення постійно розвивається; відбуваються технологічні зміни та способи організації роботи проектних команд. В епоху економіки знань більшість економічних видів діяльності вимагають всіх видів систем і IT-продуктів. Це спричиняє відповідне прискорення розробки ПЗ, водночас зберігаючи необхідну якість. Ось чому так важливо мати відповідну команду для впровадження складних та інноваційних IT-рішень.

З іншого боку, численні проектні команди в рамках одного проекту створюють проблеми в ефективній комунікації в групі та викликають необхідність оптимізації організації роботи. Для забезпечення цієї оптимізації часто використовуються інструменти та IT-продукти, призначені для управління процесами розробки програмного забезпечення.

Традиційні підходи до управління IT-проектами стали недостатніми через високу варіативність вимог і необхідність зміни організації роботи проектних команд [2, 3, 4]. Особливо це помітно в інноваційних проектах розробки програмного забезпечення для електронної комерції, де потрібна гнучкість виробничого процесу та оригінальність рішень.

Таким чином, незважаючи на підтримку процесів розробки програмного забезпечення дедалі кращими інструментами та системами, пошук оптимального використання проектних ресурсів, включаючи людські команди, все ще є серйозною проблемою.

З початку 21 століття в практиці розробки програмного забезпечення все частіше використовується Agile підхід. Література з цього питання свідчить про значне збільшення кількості успішних проектів та оптимізації використання ресурсів, необхідних для розробки програмного забезпечення, завдяки використанню гнучких методів, що постійно розвиваються. Це особливо важливо через постійні зміни вимог до програмного забезпечення, що розробляється [2]. Таким чином, традиційні методології не можуть бути успішно застосовані до численних змін, які є природними в інноваційних проектах,

оскільки вони є надзвичайно статичними та, крім першої фази, складаються з фіксованої кількості завдань.

Наступний етап розробки методології заснований на ітераціях, що передбачає, що ми дізнаємося про нові вимоги в процесі розробки – змінюються сформульовані вже вимоги та деталізуються існуючі. Однак у поточних гнучких методологіях вимоги є неповними, і ми не знаємо всіх деталей. Це також зумовлено характером інноваційних проектів [5].

У нас є набір вимог у вигляді епіків та історій користувачів. Потім, у процесі розробки програмного забезпечення, створення нових вимог стає причиною нових завдань. Іншим джерелом нових завдань впровадження є тести поточних версій, які виявляють дефекти або розкривають нові можливості впровадження. Автоматизація цього процесу генератором завдань у гнучких методологіях дозволяє створювати відповідні прирости завдань, кількість і характер яких залежить від розміру та типу проекту.

З іншого боку, ітераційне вимірювання завдань дозволяє планувати подальшу ітерацію з припущенням незмінності певних аспектів для найкращої оцінки. Тому методи планування, засновані на сумарній роботі, сьогодні мають велике практичне значення для управління виробничими процесами.

Системи управління проектами для розробки програмного забезпечення збирають багато даних про завдання та пов’язану з ними інформацію, необхідну для виконання роботи та необхідного обміну інформацією між учасниками команди проекту. Проте ще є місце для вдосконалення, коли йдеться про надання інформації, яка оптимізує управління IT-проектами. Оскільки дані та інформація, зібрані в системах, є семантично бідними, необхідно аналізувати роботи, які виконуються під час виконання завдань, щоб правильно їх класифікувати, що також дозволить оцінити їх розмір, підвищуючи таким чином якість планування.

Виокремлення та класифікація характерних типів завдань і роботи, що виконується в їх рамках, дозволить автоматично виконувати аналізи та оцінки

для створення глибокої статистики та звітів, необхідних для прийняття оптимальних прогнозних рішень. Крім того, врахування знань про поточні технологічні та бізнес-складові дасть належне уявлення про поточний стан роботи над проектом і дозволить більш оптимально планувати процес розробки програмного забезпечення.

Дана робота має на меті виявити вплив типу завдань на ефективність та вказати на ті фактори, які позитивно впливають на ефективність роботи команд розробників програмного забезпечення. Модель типів завдань була побудована на основі аналізу даних реальних програмних проектів із фінансового сектору, керованих системою Jira. Підключення моделі до системи Jira дозволяє легко отримувати дані для аналізу та збільшує її комерційний потенціал. Окремий абстрактний рівень моделі в поєднанні з виділеною базою даних підтримує можливість створення інтерфейсів до інших систем управління IT-проектами.

У роботі зроблено спробу класифікації проектних завдань, таким чином визначено необхідні витрати на завдання різного типу. У поєднанні з результатами досліджень завдань, створених у процесі виробництва та циклічних робіт, це дозволяє планувати проекти. Пов’язування типів завдань із ролями проектної команди дозволяє симулювати проектну роботу та підтримує управління проектом шляхом виявлення вузьких місць у виробничому процесі та уникнення надмірної зайнятості.

У кваліфікаційній роботі розглядаються основні поняття контексту, в якому реалізуються з проекти розробки програмного забезпечення, та розробка методології оптимального управління цими процесами. При цьому охарактеризовано принципи та практики, а також життєвий цикл проектів розробки програмного забезпечення, що важливо для можливості впровадження вдосконалень у цей процес.

Концепція моделі процесу програмування базується на гнучкому підході до керування програмним забезпеченням під час його розробки. Розширення оригінального підходу до ролей членів команди та типів завдань дозволяє

точніше планувати роботу в проекті та керувати командою проекту, включаючи виявлення вузьких місць або невикористаних ресурсів. В роботі розглядається модель процесу програмування, яка орієнтована на ручне розпізнавання типів завдань, що забезпечує ефективну підтримку планування завдань, які виконуються в інноваційних IT-проектах.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ

Розробка ПЗ в ІТ-компаніях в основному здійснюється шляхом відповідної організації роботи проектних команд. На жаль, в проектах з пошуку і створення нових рішень, де природні варіативність і незвичайність ідей і вимог, виникає постійна потреба в удосконаленні управління цим процесом. Через інноваційний характер ІТ-проектів проектні групи обтяжені високим ризиком. Це змушує шукати оптимальні методи та прийоми управління проектами, які дозволять краще контролювати та використовувати ресурси компанії. Сьогодні аналітичні методи та інструменти, вбудовані в ІТ-системи, є великою підмогою в підтримці управління проектами.

1.1 Роль гнучких методологій в сучасній індустрії розробки програмних продуктів

Традиційні методології управління проектами розробки програмного забезпечення зараз замінюються гнучкими методологіями або доповнюються ними. Розвиток методів управління проектами в ІТ-індустрії став результатом незадоволеності малою кількістю успішних проектів. Було виявлено велику проблему в управлінні інноваційними проектами, де жорсткі та сильно формалізовані традиційні методології розробки програмного забезпечення не працювали та навіть ускладнювали внесення змін та належне функціонування проектів [6] . Гнучкі методи використовуються в першу чергу через бажання зменшити кількість помилок, скоротити час, необхідний для створення готової продукції, або зменшити загальні затрати виробництва.

Однак, як зазначають автори в [7], при прийнятті рішень щодо впровадження гнучких практик важко однозначно стверджувати про більші успіхи в реалізації ІТ-проектів. Це пов’язано з різноманіттям трактувань поняття «успіх ІТ-проекту» та тим фактом, що використання однакових методологій у

різних компаніях однієї галузі призводить до різних результатів [8] . У літературі існує поширена точка зору , що ключ до успіху в управлінні проектами – це знання відповідних методів та інструментів. Однак інструментальний рівень відповідної методології управління проектами сам по собі не забезпечить його успіх, якщо його учасники не будуть належним чином застосовувати, тому що найважливішою частиною проектів розробки програмного забезпечення є люди.

Тому слід знайти відповідний баланс між жорсткими (бюджет, графік і час реалізації) і м’якими (комунікація, зміни, мотивація та компетенції) елементами проекту. Таким чином, незалежно від використовуваної методології, управління проектами тісно пов’язане з управлінням людьми, і практика показує, що більшість проблем, які впливають на успіх проекту, є результатом упущення «чисто людських аспектів» управління командою [9] .

Відповідно, ця робота має на меті виявити основні проблеми, пов’язані з управлінням персоналом для успіху IT-проекту, забезпечуючи ефективне планування завдань, що виконується в інноваційних і, отже, змінних процесах розробки програмного забезпечення.

З початку 21 століття ми стали свідками динамічного розвитку гнучких методологій [3]. Це пов'язано з адаптацією процесу управління до проектів з високим ступенем інновацій. У цих випадках жорсткі методології, такі як PRINCE2 та PMI PMBoK, не працюють через детальне та довгострокове планування, яке не в змозі врахувати майбутні зміни [4, 5]. При таких характеристиках проектів занадто високий рівень стандартизації діяльності не спрацьовує, оскільки часто трапляються безуспішні спроби встановити життєвий цикл проекту та деталізувати вимоги вже на етапі ініціації проекту.

Незважаючи на те, що такий підхід дає великий комфорт для виконавців, у випадку інноваційних проектів він не відповідає реальним потребам, які будуть відомі лише на наступних етапах реалізації. Хід роботи над інноваційним проектом виявляє необхідність неодноразової перевірки багатьох припущень,

дій і планів, тому що тільки в ході реалізації набуваються знання і фактична візуалізація розроблених рішень у вигляді кінцевого продукту.

Крім того, певні шляхи досягнення конкретних результатів виявляються неефективними лише після етапу перевірки та тестування. Слід також згадати зовнішні зміни, тобто зміни, які знаходяться поза контролем проектних команд, наприклад, зміни в правилах і правових нормах або поточній ситуації на ринку. Часто послідовна реалізація плану призводить до створення функціональних можливостей, які не будуть адаптовані до реальності або які стануть корисними лише після дорогих модифікацій. Тому в цій роботі зроблено спробу класифікувати характерні типи завдань у проектах розробки програмного забезпечення, які дозволять ефективніше планувати та адаптувати необхідну роботу до динамічно мінливої реальності.

Усі фактори, які ускладнюють або навіть унеможливлюють точне визначення та опис результатів проекту на етапі його визначення, є результатом наступних елементів, які виникають при створенні програмного забезпечення (особливо інноваційного програмного забезпечення) :

- замовники та користувачі не впевнені, якого результату вони очікують, і мають труднощі з формулюванням вимог;
- багато деталей і рішень будуть відомі лише в ході реалізації проекту;
- деталі впровадження інноваційних проектів на стадії планування є нездійсненними;
- спосіб мислення змінюється під час розробки програмного забезпечення;
- зовнішні сили (такі як продукти або послуги конкурентів) призводять до змін або розширення вимог.

Крім того, реалізація інноваційних ІТ-проектів пов’язана з ризиком інших порушень. Автор роботи [10] перераховує кілька основних проблем, які можуть вплинути на будь-який проект, а саме:

- управління спеціальними вимогами;

- неоднозначне і нечітке спілкування;
- крихка архітектура системи;
- надзвичайна складність і невиявлена непослідовність у вимогах, дизайні та реалізації;
- недостатнє тестування;
- суб'єктивна оцінка стану проекту;
- безконтрольне внесення змін;
- недостатня автоматизація.

Вищевказані фактори та елементи управління реалізацією проекту та неможливість їх вирішення відповідно до традиційного підходу безпосередньо сприяли розробці методів, що дозволяють зробити традиційні методології більш гнучкими, і, як наслідок, підготували ґрунт для Manifesto for Agile. Розробка програмного забезпечення, яка оголосила в 2001 році принципи гнучкої методології розробки програмного забезпечення [11]:

- окремі люди та їх взаємодія (а не процеси та інструменти);
- працюючі продукти, тобто програмне забезпечення (більш ніж повна документація);
- співпраця з клієнтом (більше, ніж узгодження договору);
- реагування на зміни (а не дотримання плану);
- задоволеність клієнтів є найвищим пріоритетом, і її слід досягати шляхом ранньої та постійної доставки цінного програмного забезпечення;
- змінність вимог стосується як нових вимог, так і вимог, що змінюються під час реалізації проекту; адаптивний процес розробки програмного забезпечення здатний завчасно встигати за змінами;
- часта поставка робочого програмного забезпечення в періоди від кількох до кількох тижнів, причому бажано коротші часові рамки;
- спілкування, яке має здійснюватися безпосередньо;
- працююче програмне забезпечення, оскільки це найважливіший показник прогресу виконання робіт у проекті;

• адаптивність розробки програмного забезпечення полягає в здатності підтримувати належний темп роботи під час виконання проекту всіма членами команди проекту та адаптувати продукт (програмне забезпечення) до вимог, які часто змінюються.

Принципи адаптивного підходу до управління проектами розробки ПЗ вказують напрямок для проектних команд, тоді як для реального виконання робіт необхідна конкретна практика. Структура процесу та специфічні практики створюють мінімальну гнучку структуру для самоорганізації команд. ІТ-інструменти необхідні для прискорення розробки програмного забезпечення та зниження витрат. Контракти мають вирішальне значення для розвитку відносин клієнт-постачальник. Документація підтримує спілкування.

Однак ключовим питанням є надання команді проекту зворотнього зв’язку, щоб відповісти на питання про те, де зараз знаходиться команда в процесі розробки програмного забезпечення. Це можливо завдяки ітераціям і поступовим приростам властивостей продукту в життєвому циклі програмного забезпечення. Суть ітераційного процесу полягає в частій доставці робочих частин програмного забезпечення (послідовних приростів), які реалізують вибрані набори функцій, які разом складають корисність кінцевого продукту.

Ітеративний цикл розробки програмного забезпечення веде до стилю управління, коли довгострокові плани нечіткі чи незавершені, тоді як стабільний план може бути створений на короткий проміжок часу. Ітераційна та інкрементальна розробка програмного забезпечення призводить до абсолютно нових відносин із бізнес-клієнтом та інших принципів функціонування команди проекту.

1.2 Характеристики гнучких методологій розробки програмного продукту

Концепція циклу розробки ітераційного та інкрементального програмування як засобу вирішення дилем епохи електронної економіки призвела до створення нових методологій для управління IT-проектами. Ці методології, які називають гнучкими, не відриваються повністю від документоорієнтованих формалізованих традиційних методологій, але мають специфічні особливості (адаптовані до вимог сучасних проектів розробки програмного забезпечення) [12]:

- адаптивні, не прогнозуючі, традиційні методології не справляються з частими змінами вимог, а гнучкі принципово їх приймають;
- орієнтація на людей, а не процес;
- творчий стиль роботи.

Через загрозу збоїв організації все частіше впроваджують плани цифрової трансформації, щоб не відставати від темпів зростання бізнесу. Гнучка розробка програмного забезпечення відіграє величезну роль у цьому процесі. Багато цифрових робочих процесів, які використовуються сьогодні, базуються на гнучких принципах. Завдяки гнучкій масштабованій IT-інфраструктурі хмарні обчислення розвиваються відповідно до потреб гнучкої розробки програмного забезпечення.

Концепція DevOps усуває традиційну різницю між розробкою програмного забезпечення та операціями. Програмне забезпечення використовується як інструмент у впровадженні SRE–DevOps та системному управлінні та автоматизації операційних обов’язків. Методології CI/CD підтверджують, що програмне забезпечення буде часто змінюватися, і надають інструменти, які допомагають розробникам швидше доставляти новий код [13].

Гнучкі методології мають різні форми, щоб задовольнити потреби будь-якого проекту . Незважаючи на те, що гнучкі підходи відрізняються, усі вони

базуються на ключових ідеях, які містяться в гнучкому підході. З цієї причини будь-яка структура або поведінка, яка відповідає цим принципам, називається гнучкою. Незалежно від конкретних гнучких підходів, які команда вирішує застосувати, переваги гнучкої методології можуть бути повністю реалізовані лише завдяки співпраці всіх залучених сторін. В останні роки з'явилася велика кількість гнучких методологій управління проектами розробки програмного забезпечення. Найбільш популярними є наступні.

- Канбан. Фраза «канбан» (японська) перекладається як «візуальна дошка або вивіска» і асоціюється з ідеєю «точно вчасно». Концепція Kanban поступово знайшла свій шлях до гнучких команд розробників. Такий підхід розвиває управління проектами за допомогою візуальних методів. Дошка Kanban, розділена на стовпці, щоб проілюструвати процес розробки програмного забезпечення, використовується для контролю над проектами.

Команди отримують вигоду від кращої видимості, оскільки вони можуть відстежувати свій прогрес на кожному етапі розробки та можуть планувати майбутні завдання для доставки продукту за розкладом. Щоб гарантувати, що члени команди завжди мають безперебійний робочий процес і знають про відповідний етап розробки, цей метод потребує комплексного спілкування та прозорості.

- Scrum. Гнучкий підхід до розробки Scrum, який представлений кількома циклами розробки, є одним із найвідоміших прикладів гнучкої методології. Scrum розбиває процеси розробки на одиниці, відомі як «спринти», схожі на Kanban. Максимально збільшуючи та приділяючи час розробці кожного спринту, одночасно можна керувати лише одним спринтом.

Послідовні результати, підкреслені методами Scrum і Agile, дозволяють дизайнерам змінювати пріоритети таким чином, щоб будь-який неповний або затриманий спринт привертав більше уваги. Щоденний міт – це місце, де координуються дії для розробки найкращої стратегії спринту; Scrum-команда має ексклюзивні дизайнерські ролі, такі як Scrum-майстер і власник продукту.

- XP (екстремальне програмування). Методологія екстремального програмування (XP) приділяє велику увагу співпраці, діалогу та зворотному зв’язку. Це підкреслює постійне вдосконалення та задоволеність клієнта. Цей підхід використовує спринт або короткі цикли розробки, схожі на Scrum. Він створений командою для створення високоефективної та продуктивної атмосфери. Техніка екстремального програмування дуже корисна в ситуації, коли потреби клієнтів постійно змінюються. Це заохочує розробників приймати зміни до вимог замовника, навіть якщо ці вимоги з’являються на пізній стадії процесу розробки. У екстремальному програмуванні проект оцінюється з самого початку шляхом збору вхідних даних, які підвищують продуктивність системи. Крім того, він пропонує швидкий спосіб задовольнити будь-які вимоги клієнтів.
- Сімейство Crystal Clear, Концепція, розроблена Алістером Кокберном, експертом з об’єктно-орієнтованого дизайну. Кожен клас проекту може мати різну методологію (метод Crystal Clear). Crystal – це набір менших підходів гнучкого програмування, які включають Crystal Yellow, Crystal Clear, Crystal Red, Crystal Orange тощо. Його вперше представив пан Алістер Кокберн, одна з ключових фігур у створенні Agile Manifesto for Software Development. Кожен з них має унікальну структуру, яка відрізняє його від інших на основі таких змінних, як критичність системи, розмір команди та пріоритети проекту.

Тип гнучкого підходу Crystal вибирається в залежності від критичності проекту або системи. Crystal прагне до своєчасної доставки продуктів, регулярності, скороченого адміністрування з високим рівнем взаємодії з користувачем і задоволеності клієнтів, подібно до інших гнучких підходів. Сімейство Crystal, яке заслужило звання «Найпростіші шляхи гнучкої методології», пропагує ідею, що кожна система або дизайн є унікальними та потребують застосування різних практик, процесів і принципів для отримання найкращих результатів.

- Адаптивна розробка програмного забезпечення, широка адаптивна методологія, розроблена Джимом Хайсмітом.

• Динамічна розробка системи (DSDM). Цей метод динамічної розробки систем був створений, щоб задовольнити потребу в уніфікованому галузевому статуті для швидкої доставки програмного забезпечення. Процес розробки програмного забезпечення можна планувати, запускати, керувати та масштабувати за допомогою комплексної структури, наданої DSDM, яка підтримує цю якість і безперебійність. Своєчасна доставка ніколи не може бути поставлена під загрозу, і що зміни в дизайні завжди слід очікувати. Це переконання ґрунтується на восьми принципах і бізнес-методології.

• Lean development, «ощадлива» розробка програмного забезпечення. Основною ідеєю підходу є усунення втрат, які розуміються як елементи, які не додають вартості продукту. Метою такої дії є якнайшвидша доставка готового продукту замовнику. Розробка ощадливого програмного забезпечення базується на цінностях і принципах адаптивного управління проектами. Термін «ощадлива розробка програмного забезпечення» розглядають Мері Поппендік і Том Поппендік , які у своїй книзі «Бережлива розробка програмного забезпечення: гнучкий інструментарій» представили, серед іншого, сім основних принципів ощадливого управління та набір із 22 методів, що підтримують підхід.

Гнучкі методології управління – це група методологій, які характеризуються адаптивним і варіативним підходом до управління проектами розробки програмного забезпечення. Крім того, гнучкі методології були розроблені набагато раніше, ніж сам гнучкий маніфест. Перші роботи з адаптивних методів управління проектами відносяться до 1980-х років (прикладом є методологія швидкої розробки додатків), а поняття гнучких методологій було введено в середині 1990-х років .

Ідея часових рамок – це чітко визначений процес, присвячений контролю розробки програмного забезпечення на найнижчому рівні в ітераційному циклі з кількома точками перегляду. Огляди допомагають забезпечити якість та ефективність розробки програмного забезпечення. Вчасно надаючи програмне

забезпечення на найнижчому рівні, забезпечується своєчасне виробництво на найвищому рівні (тобто на рівні проекту).

Основним принципом плану проекту є підготовка графіка запланованих приростів і, в їх межах, запланованих часових рамок, які створять повний графік проекту, здатний змінюватися з появою нових вимог. Використання техніки часових рамок разом із технікою пріоритезації забезпечує відсутність затримок у реалізації проекту та поставку готового програмного забезпечення, яке відповідатиме бізнес-цілям протягом заданого часу.

Проекти з високим ступенем інновацій дуже важко включити в повний графік і обсяг робіт. Тому адаптивні методології описують функціональні можливості (тобто незалежні елементи підсистеми), які в наступних випусках можна швидко змінити та передати для впровадження. Гнучкі методології, на відміну від традиційних методологій, виключають валідність довгострокового планування. Тому плани спекулятивні, а не детерміновані. Це дозволяє адаптуватися до всіх типів змін, які виникають під час реалізації проекту. Крім того, відмінною рисою гнучкої методології є сильний акцент на співпраці та інтеграції команди проекту, оскільки лише в цьому випадку забезпечується плавний потік інформації та ефективна комунікація. Загальна схема життєвого циклу проекту у випадку гнучких методологій базується на п’яти фазах, зазначених в [18]:

1. Створення бачення проекту шляхом визначення його обсягу та принципів співпраці в рамках команди проекту.

2. Заплановані припущення шляхом визначення функціональних елементів для продукту, створення обмежених у часі планів ітерацій та основних етапів самого проекту.

3. Розвідка, тобто швидкий старт шляхом надання користувачеві функціональних елементів і впровадження методів виробництва, які мінімізують витрати на зміни, включаючи створення команди проекту, яка адаптується та співпрацює.

4. Адаптація, тобто оцінка продукту, процесу, проекту та команди проекту, а потім корекція існуючих планів і практик проектування.

5. Закриття проекту шляхом створення бази даних досвіду для наступного проекту.

Методології гнучкого управління проектами вимагають належного рівня зрілості проекту для організацій і проектних команд. Крім того, гнучкі методи продовжують удосконалюватися, щоб найбільш ефективно відповідати потребам управління все ще інноваційними проектами розробки програмного забезпечення. Отже, існує так багато методів, які все ще шукають нові, більш досконалі рішення; хоча, безсумнівно, вони вже здаються краще адаптованими, ніж класичні методи, до динамічно мінливих проектних середовищ. Реалізація на основі ітерації дозволяє ефективно адаптувати сигнали, які надходять як зсередини проекту, так і із зовнішнього середовища.

Класифікація конкретних типів завдань у проектах розробки програмного забезпечення дозволить визначити вимоги, необхідний час та витрати, які будуть потрібні для їх реалізації. Це дозволить більш ефективно працювати над подальшими методами адаптивного проектного планування. Пов’язування типів завдань із ролями команди проекту дозволить імітувати роботу над проектом, що підтримуватиме управління проектом шляхом виявлення вузьких місць у процесі розробки програмного забезпечення.

Крім того, це дозволить уникнути надмірної зайнятості та дозволить підтримувати швидке моделювання «що, якщо». При цьому запровадження класифікації завдань за бізнес- і технологічною складовою у поєднанні з моделлю компетенцій співробітника дозволить автоматично підбирати склад проектної команди та оптимально керувати командою. Це має найвищий пріоритет у гнучких методологіях. Це має велике значення для оптимізації управління процесом розробки програмного забезпечення та призводить до розробки досконалих методів у відповідь на динамізм змін реального світу. Він матиме реплікативні та прогнозні можливості для планування проекту, його

моделювання під час змін і виявлення вузьких місць протягом усього процесу. У цій статті були проведені експерименти на багатьох реальних IT-проектах, щоб класифікувати типи завдань і спробувати знайти зв’язок між характером запланованого проекту та специфікою та кількістю цих завдань.

РОЗДІЛ 2. КЛАСИФІКАЦІЯ ТИПІВ АРТЕФАКТІВ У ГНУЧКИХ ПРОЕКТАХ

2.1 Дослідний набір даних

Модель типів завдань була побудована на основі аналізу даних реальних програмних проектів із фінансового сектору, керованих системою Jira. Таблиця 2.1 містить зведені дані цих проектів. У чотирьох проектах команда працювала за традиційними методологіями, до яких впроваджувалися все нові й нові елементи гнучкого підходу. У двох проектах команди працювали за підходом Scrum. Це досить великий набір даних, який дозволяє аналізувати явища, пов'язані з сучасними виробничими процесами. Загальна кількість релізів проекту перевищує 30 тис., що дає можливість використовувати методи штучного інтелекту. Загальна тривалість проектів становить приблизно 22 роки, що не означає, що мова йде про історичні записи. Проектні групи працювали над більшістю проектів паралельно.

Таблиця 2.1 – Проекти, дані яких використовувалися для дослідження типів завдань.

Проект	**Методологія**	**Число артефактів**	**Затрати, людино/годин**	**Тривалість проекту**		**Розмір команди**		
				років	**місяці**	**Мін**	**Середнє**	**Макс**
P1	Гібрид	10773	67444	8,0	96,3	2	14	26
P2	Гібрид	2492	17063	3,6	43,6	1	12	25
P3	Гібрид	7754	41530	3,2	37,9	1	19	31
P4	Гібрид	1212	7453	2,6	30,6	1	8	21
P5	Scrum	5466	55354	3,4	40,9	6	22	39
P6	Scrum	3949	27397	1,9	22,8	1	23	41
	Всього	31646	216241	22,7			98	

У Jira команди використовують артефакти типу issue для опису та відстеження конкретних завдань, які потрібно виконати. Артефакти є основними будівельними блоками проектів. Артефакт може бути певного типу. Найпоширенішим видом артефакту є завдання (таск), яке на практиці використовується для багатьох цілей. Другим за поширеністю типом є помилка (дефект, баг), яка описує дефекти, виявлені в розробленому програмному забезпеченні, і роботу, необхідну для їх усунення. Повний перелік типів, що використовуються в проектах, із їх частотою наведено на рисунку 2.1 .

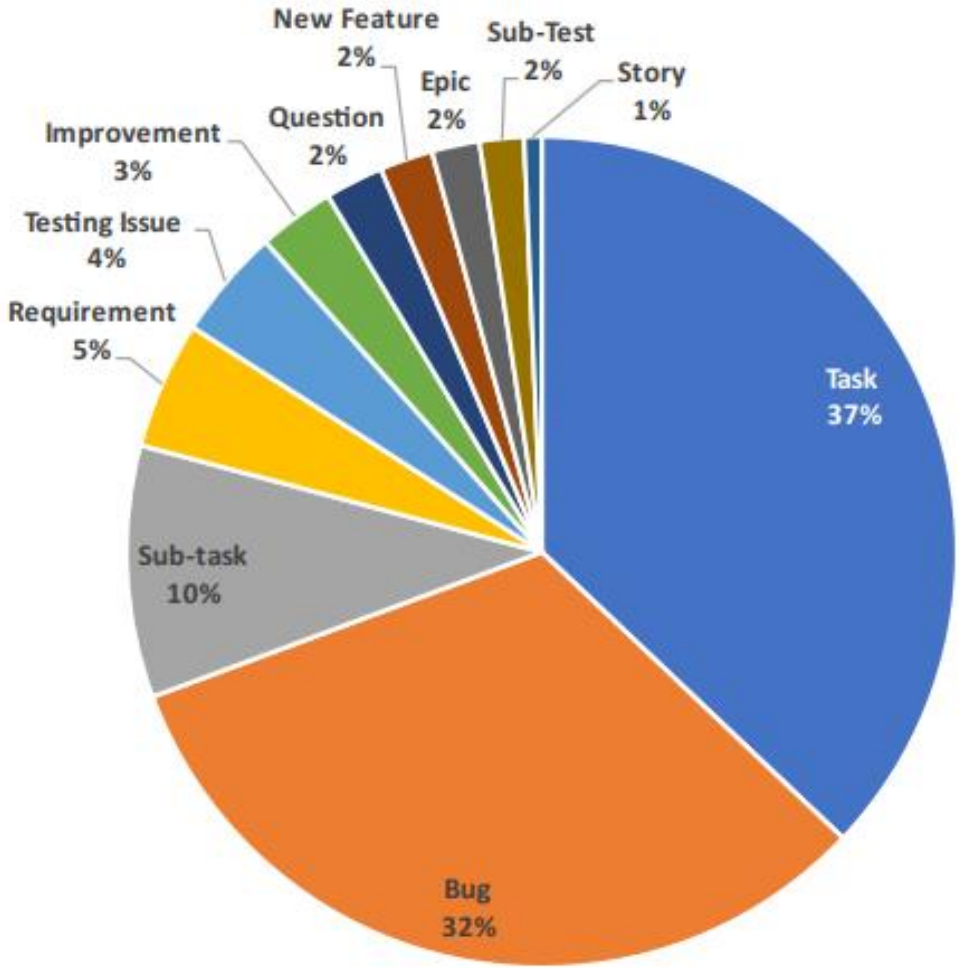

Рисунок 2.1 – Частота появи різних типів артефактів
у досліджуваних проектах

Найпоширенішими типами проблем є завдання, помилка та підзавдання. Всі типи артефактів мають певні атрибути. Зокрема, саммарі (короткий опис)

визначає ідею дій, які необхідно виконати. Поле опису містить детальний опис роботи, яку потрібно виконати.

Суттєвою особливістю артефакту є його стан. Відстеження змін стану артефакту дозволяє оцінити виконану роботу. Робочий процес описує значення поля стану та дозволені переходи між ними. На рисунку 2,2 показано типовий спрощений робочий процес, який використовується в проаналізованих проектах.

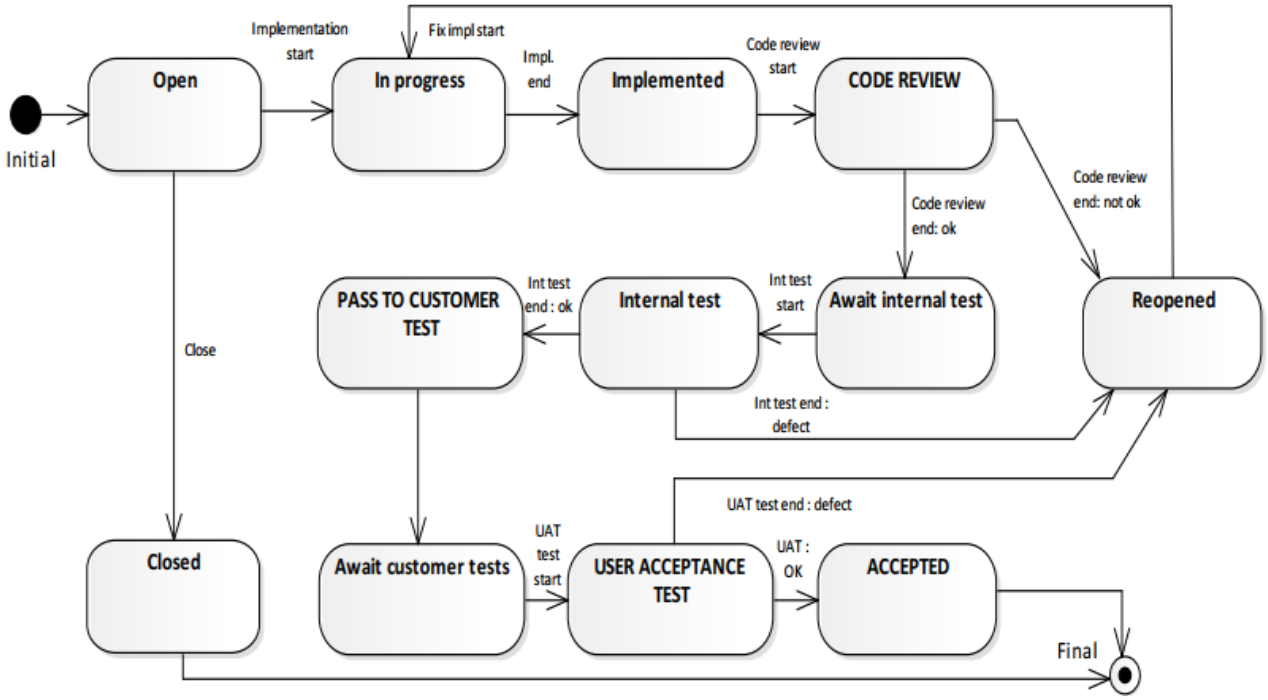

Рисунок 2.2 – Типовий спрощений робочий процес, який використовується в проектах

Як показано на рисунку 2.2, переходи між станами артефакту відповідають виконанню основних дій із впровадження нових функцій системи або виправлення помилок. Наприклад, методологія DevOps (розробка та операції) допомагає налагодити співпрацю між розробниками та операторами для автоматизації безперервної доставки нового програмного забезпечення, що, як очікується, сприятиме скороченню циклу розробки та створенню програмного забезпечення високої якості [15]. Іншою розробкою DevOps є концепція розробки, безпеки та операцій (DevSecOps), яка в той же час призначена для

інтеграції методів безпеки з процесом розробки програмного забезпечення, де вбудовані заходи безпеки для забезпечення цілісності та доступності програми.

Цінною особливістю системи Jira є збереження історії змін вартості елементів емісії, включаючи стан. Зберігання історії підтримує відстеження виконання проблеми. Не всі питання використовують стан для відстеження роботи. Система Jira дозволяє співробітникам реєструвати робочий час, присвячений роботі над питанням. Для деяких видів робіт немає необхідності відстежувати їх стан, оскільки це повторювані роботи, які виконуються за потреби. У цьому випадку достатньо можливості реєстрації робочого часу.

Представлений підхід базується на точному розподілі процесу розробки на дії та ролі відповідно до методології RUP, адаптованої до гібридних та гнучких процесів [16]. У ряді наукових робіт, наприклад [17, 18] описувалась агентно-об’єктна модель виробничого процесу (AGOMO), яка спочатку була використана для оцінки зрілості процесів RUP, а потім для планування гібридних процесів Water–Scrum–Fall.

Як ми показали в попередньому розділі, типів завдань Jira недостатньо для чіткого визначення мети та типу роботи. Модель створена для того, щоб заповнити прогалину, яка заважає пов'язати виконану роботу зі складом і компетенціями команди проекту. Поєднання цих двох напрямків дозволить провести більш точний кількісний аналіз робіт проектів і виявити причини спостережуваних явищ. Це спосіб оптимізувати ефективність процесів розробки та підвищити рівень зрілості проектних команд та організацій.

Проекти програмного забезпечення складаються із завдань, які команда проекту виконує для створення IT-системи. Завдання представляє проблему Jira. Кожне з виконуваних завдань має чітко визначену мету. Цією метою може бути, наприклад, реалізація вимоги, тестування компонента або всього модуля, адміністрування середовища або керування проектом; це визначає його тип. Члени команди проекту виконують завдання. Ролі визначають компетенцію та

відповідальність тих, хто виконує завдання. Одна людина може мати багато ролей; одну роль може мати багато людей.

Завдання реалізації функцій системи вимагає виконання деяких суттєвих підзадач. Вони включають реалізацію, тобто створення вихідного коду компонента, перегляд коду, створення модульних тестів, тестування та перевірку функцій, а також приймальні тести. Такі завдання відповідають проблемі, стан якої змінюється відповідно до робочого процесу, показаного на рисунку 2.2. Завдання реалізації, які складаються з підзадач, у моделі називаються завданнями зі збереженням стану.

Кожне завдання та підзавдання містить кількість робочих годин членів команди проекту. Типи завдань і підзавдань дозволяють асоціювати зусилля з ролями членів команди проекту. Вони допомагають розпізнати вузькі місця або надмірну зайнятість у завершених IT-проектах і уникнути їх під час планування проекту.

Артефакти, стани яких не змінюються протягом проекту, представлені в моделі повторюваними завданнями. Їх назвали повторюваними, тому що одне завдання цього типу може бути виконане за потреби протягом усього проекту. Наприклад, це може бути завдання з управління розробкою проекту, яке виконує адміністратор, коли виникають дефекти або коли потрібно налаштувати нові програмні компоненти. Для повторюваних завдань можна розрахувати середню трудомісткість за період і на цій основі припустити, якими є постійні витрати на проект. По-справжньому повторювані завдання – це, наприклад, зустрічі, такі як щоденні зустрічі, семінари з клієнтами або зустрічі керівної групи.

Було проаналізовано типи завдань (артефактів) досліджуваних проектів за типом. Результати у вигляді графіків, що показують відсоток і розміри завдань із збереженням стану та повторюваних завдань, показані на рисунку 2.3. На діаграмі ліворуч показано співвідношення кількості завдань із збереженням стану до повторюваних завдань у проектах. Діаграма праворуч показує співвідношення зусиль завдань із збереженням стану та повторюваних завдань.

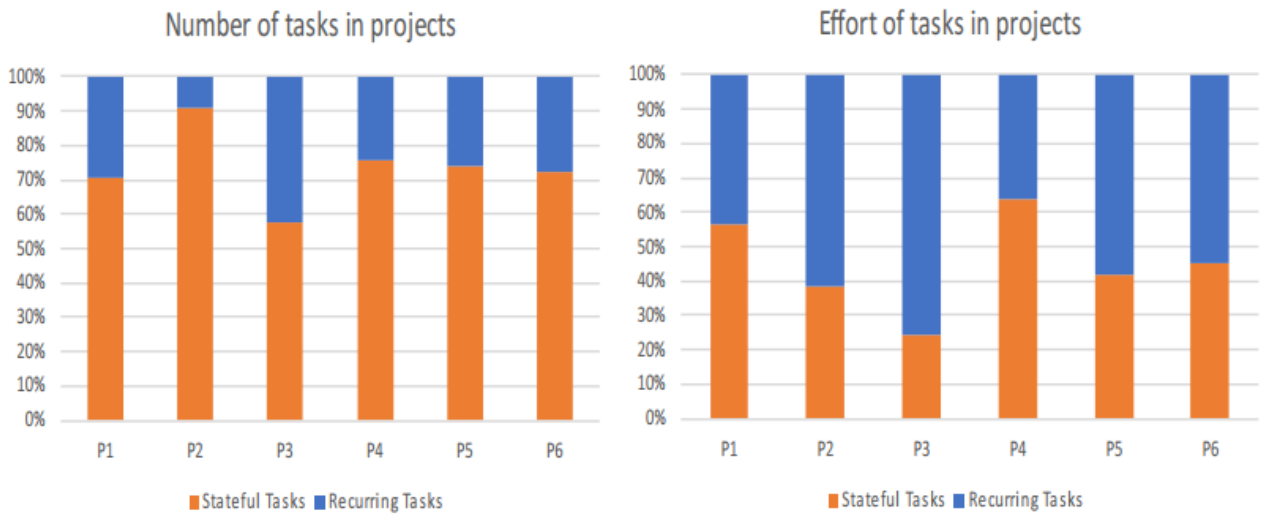

Рисунок 2.3 – Кількість і зусилля завдань із збереженням стану та повторюваних завдань у досліджуваних проектах

Показовим є випадок проекту P2, де кількість завдань із збереженням стану перевищує кількість повторюваних завдань (більше 90%), а трудомісткість цих завдань становить трохи більше 40%. Це дуже цікаво, тому що завдання з підтримкою стану відповідають за реалізацію функцій побудованої системи та за усунення виявлених у ній дефектів. З точки зору розробника, це навантаження має бути, мабуть, найбільшим. З точки зору дослідника процесів розробки програмного забезпечення, це явище викликає велику цікавість. Щоб його задовольнити, необхідно насамперед чітко визначити типи завдань у проектах, які реалізує дана стаття на основі фактичних та реалізованих проектів із практичної діяльності.

2.2 Результати опрацювання даних

Класифікація типів завдань базується на розподілі завдань на поточні та повторювані. За визначенням, завдання з збереженням стану являють собою роботу, пов'язану з впровадженням системних компонентів і виправленням дефектів. Завдання з підтримкою стану обробляються алгоритмом підзадач. За решту робіт відповідають повторювані завдання. Повторювані завдання можна

класифікувати на основі короткого та розширеного текстового опису, включеного до випуску. Спочатку ці завдання класифікувалися вручну. Згодом на основі пошуку за ключовими словами були створені прості алгоритми класифікації. Повторювані завдання також розбиваються на підзавдання, які відповідають зареєстрованій роботі.

На рисунку 2.4 зображено діаграму діяльності алгоритму класифікації. Алгоритм спочатку перевіряє, скільки разів проблема змінювала свій стан; якщо принаймні тричі (більше, ніж відкритий і закритий), робочий процес є основою для поділу завдання на підзавдання. Алгоритм розділення намагається створити підзавдання (наприклад, розробка, тестування, перевірка коду). Для цього він використовує значення стану до та після, час зміни, роль особи та зареєстровані роботи, а потім призначає виконану роботу створеному підзавданню. Результатом є завдання зі збереженням стану.

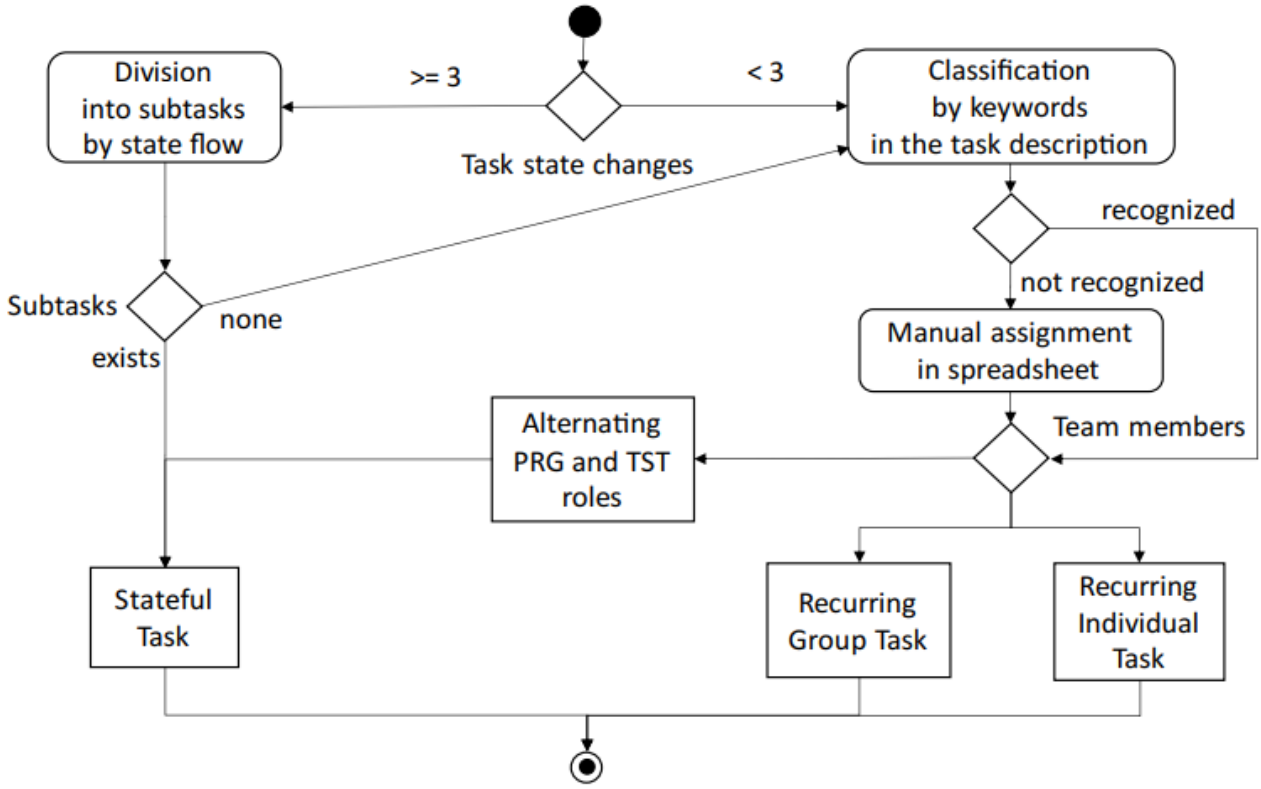

Рисунок 2.4 – Алгоритм класифікації артефактів

Якщо завдання не змінило стан або алгоритм не виявив підзадач, тип завдання визначається шляхом пошуку за ключовими словами в текстовому описі. Якщо це не вдається, завдання надходять до електронної таблиці, де вони класифікуються вручну. Потім алгоритм перевіряє, хто працював над завданням. Якщо багато людей працювали над завданням в один день, це групове завдання, що повторюється (наприклад, стендап та інші зустрічі).

Якщо одна особа працювала над завданням в один день, це одинарний артефакт, що повторюється. Спосіб поділу повторюваного завдання на підзадачі залежить від індивідуальної/групової класифікації. Якщо програміст і тестер по черзі виконують завдання, воно має статус; алгоритм розділяє їх на підзавдання відповідно до ролей членів команди.

Алгоритм розбиття завдань із збереженням стану на підзадачі дуже складний через багато способів, якими користувачі використовують Jira для роботи над артефактом проекту. Пошук за ключовими словами не є дуже точним, а підтримка його ручною класифікацією унеможливлює використання класифікації на практиці. Тому планується замінити ці рішення на класифікацію NLP (Natural Language Processing – обробка природної мови). Однак у ранжируванні вручну є кілька хороших моментів. Під час роботи над алгоритмом процес ручного аналізу повторюваних завдань виявив понад 50 типів завдань, які були об’єднані в три групи, що містять 14 основних типів завдань.

Завдання із збереженням стану складаються із завдань із впровадження нових функцій і виправлення помилок, про які повідомляють клієнти та тестувальники. Періодичні завдання поділяються на три основні групи: виконання, зустрічі та організаційні завдання. Повний перелік основних типів завдань наведено в таблиці 2.2. Типи завдань, наведені в таблиці, утворюють ієрархічну структуру, поділену на типи та групи.

Алгоритм класифікації завдань – класифікація завдань, які почергово виконуються програмістами та тестувальниками, як завдань із збереженням стану – збільшив кількість і зусилля завдань із станом. Оновлена версія графіків

на рисунку 2.3 включена в рисунок 2.5. Зміни значні. Наприклад, для проекту P3 відсоток завдань із збереженням стану зріс з 58% до 66%, а відсоток завдань із збереженням стану зріс з 24% до 43%.

Таблиця 2.2 – Типи завдань із збереженням стану та повторюваних завдань

Kind	Group	Type
Stateful	DEV requirement, function, feature	DEV-PRG: analysis, design, programming
		DEV-TST: testing, verification
	BUG-DEV defect from internal tests	BUG-DEV-PRG: programming
		BUG-DEV-TST: testing, verification
	BUG-CLI defect found by customer	BUG-CLI-PRG: programming
		BUG-CLI-TST: testing, verification
Recurring	R-DEV development	A&D: analysis and design
		BLD: build versions, find the causes of errors, create scripts
		CFG: software configuration
		CUST: training, technical support, commuting
		MIGR: data migration from previous systems
		RQM: requiremets engineering RVW: code review
		TST: subsystem, module tests, manual tests
	R-MEET meetings	M-CUST: other customer meetings
		M-DEV: development team meetings with external teams
		M-INT: development team internal meetings
		M-STDNP: daily stand-up meetings
		M-WRK: requirements workshop with the customer
	R-ORG organizational	ADM: administration of environments and servers
		PM: project management

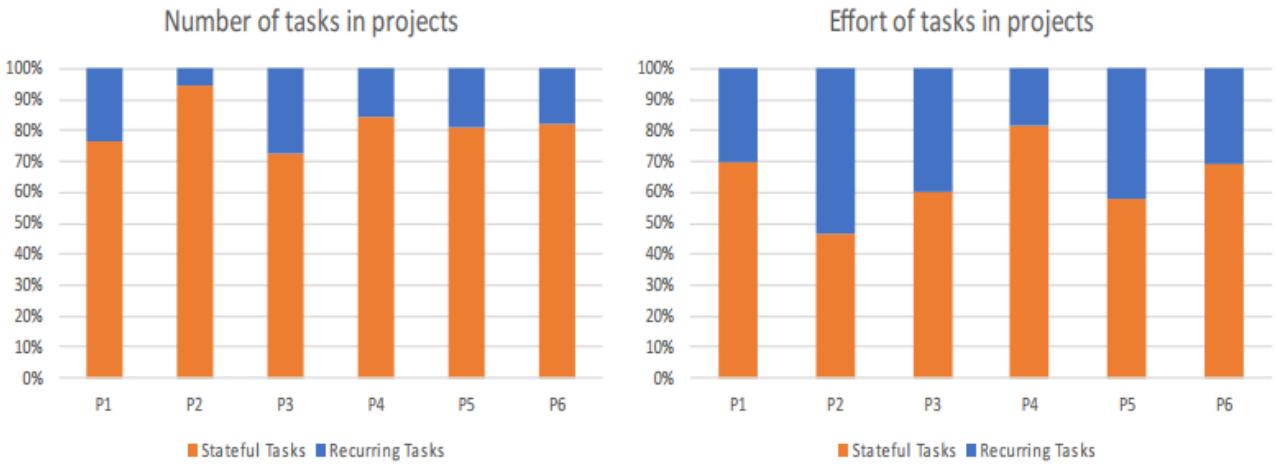

Рисунок 2.5 – Оновлена діаграма кількості та трудомісткості завдань

Розробка типів завдань та алгоритму автоматичної класифікації досліджуваних проектів дозволила провести детальний аналіз завдань досліджуваних проектів. нижче, ми наводимо діаграми зусиль (див. рис. 2.6) для шести груп типів завдань у досліджуваних проектах.

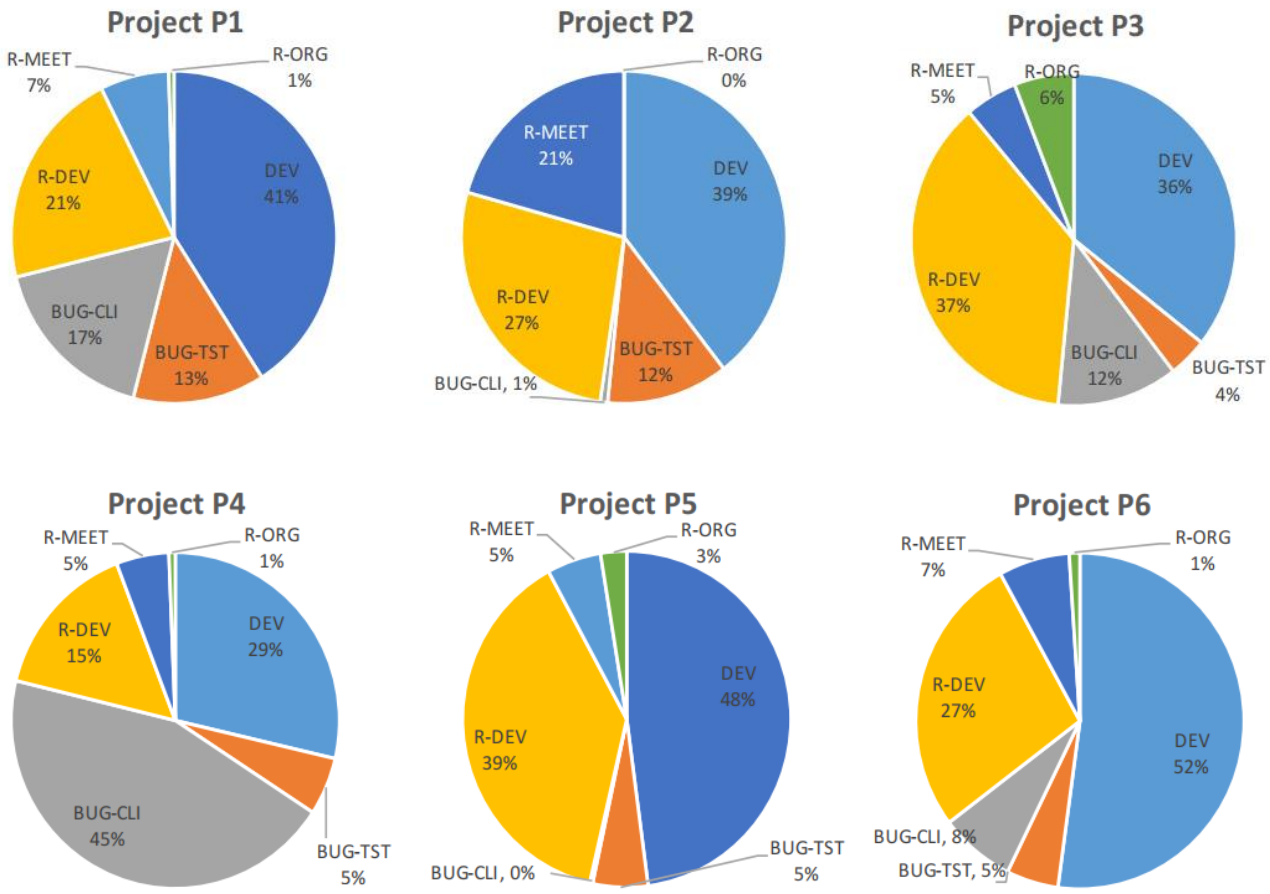

Рисунок 2.6 – Трудомісткість усіх шести типів артефактів в досліджуваних проектах

Проект P1 реалізовано за гібридною методологією. Це найдовший проект серед досліджуваних, тривалість якого перевищує 8 років. Проект пройшов через багато етапів впровадження, доставки та обслуговування. Тому значення та співвідношення зусиль у проекті були усереднені. Вони можуть служити еталоном у порівнянні з іншими гібридними та традиційними конструкціями.

Проект P2 виконано за гібридною методологією. Характеризується великою кількістю годин, які використовуються для проведення різного роду зустрічей. З іншого боку, дуже невеликий обсяг роботи в області управління

свідчить про те, що проектом, можливо, керували колективно. Дуже мало роботи, пов’язаної з усуненням дефектів, виявлених клієнтами, свідчить про те, що час, витрачений на зустрічі, був витрачений добре.

Проект P3 здійснювався за гібридною методологією. Він має високі адміністративні та управлінські витрати. Цікавим є співвідношення ремонтних дефектів, виявлених замовниками, до ремонтних дефектів, знайдених тестувальниками. В інших проектах такої тенденції немає, окрім хіба що проекту P4. Це може свідчити про неточно визначені вимоги або складний контакт із замовником.

Проект P4 реалізовано за гібридною методологією. Розподіл навантаження завдань в проекті P4 відрізняється від інших дуже великим обсягом роботи з усунення виявлених замовниками недоліків. Вартість цієї роботи в півтора рази більша, ніж виконання завдань із впровадження, і в п’ять разів перевищує вартість усунення дефектів, виявлених командою розробників. Ситуація схожа на проект P3, тільки витрати на управління та розробку значно нижчі. Можливо, проект P4 знаходиться на етапі обслуговування проекту з великою кількістю дефектів.

Проект P5 реалізовано за методологією Scrum. На високому рівні абстракції, отриманому за допомогою групового аналізу типу завдання, не видно різниці між цим проектом і гібридними проектами. Важливим є брак ресурсів для усунення недоліків, про які повідомляють клієнти. Цілком можливо, що проект не увійшов у фазу реалізації замовником і не був запущений у виробництво.

Проект P6 також був реалізований за методологією Scrum. Більше половини роботи було присвячено реалізації продукту. Продукт, імовірно, було доставлено клієнту, про що свідчить 8% зусиль щодо усунення дефектів, про які повідомили клієнти. Значну частку в роботі над проектом займають зустрічі, що узгоджується з методологією Scrum. Витрати на підтримку та управління проектом невеликі.

Графіки загальних трудомісткостей за групами типів завдань дають дуже синтетичне уявлення про проекти. Ми можемо глибше поглянути на відмінності між проектами, зосередивши увагу на докладних зусиллях циклічних типів завдань. На рисунку 2.7 показано радіолокаційні діаграми повторюваних зусиль, показаних у відсотках від загальних повторюваних зусиль. Тут показано деталі групи R-DEV з рисунка 2.6. Ліва частина графіка показує трудомісткість зустрічей, співпраці з клієнтом, адміністративної роботи та менеджменту. Права частина діаграми показує витрати на повторювані завдання розробки. Діаграми відрізняються одна від одної, щоб відобразити характеристики проектів. Діаграми показують «відбитки» проектів, що дає змогу визначити їх детальні характеристики та порівняти між собою.

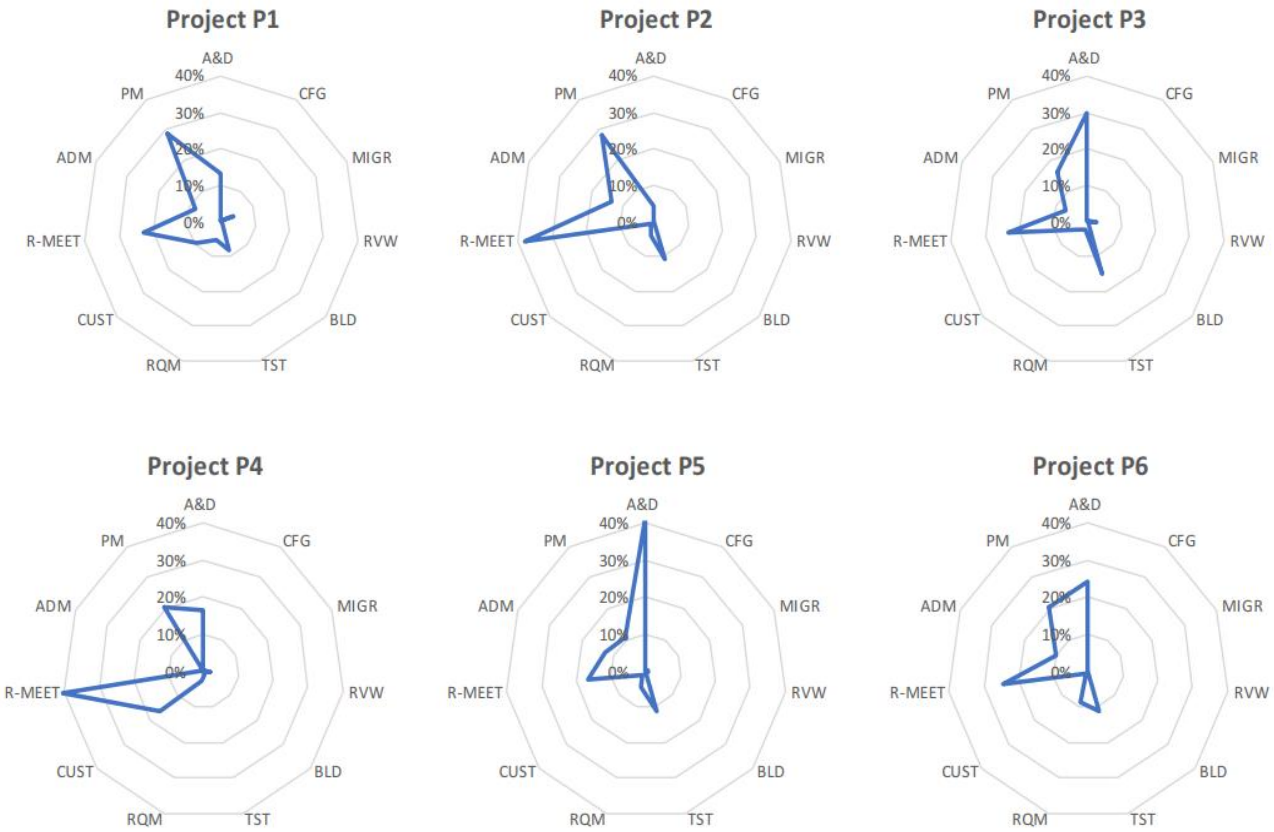

Рисунок 2.7 – Структура витрат на повторювані завдання в досліджуваних проектах

У більшості діаграм ліва сторона організації та управління домінує над правою стороною розробника. Проекти P1, P2, P3 і P6 дотримуються схожої

моделі в графіку повторюваних робіт, що вказує на більші витрати на зустрічі та управління, ніж на роботу з розробки. Аналіз і дизайн відіграють велику роль у проектах P3, P5 і P6.

Порівняння деталей повторюваної роботи в проектах дозволяє визначити мінімальне, середнє та максимальне значення повторюваної роботи, що дасть можливість використовувати лінгвістичні змінні в роботі для планування проекту. На основі статичних підсумкових графіків можна визначити характер проекту та порівняти проекти між собою. Також важливо визначити площу витрат повторюваних завдань у проектах і визначити їх мінімальне, середнє та максимальне значення. Це важливо для планування проекту. Введені типи завдань можуть бути корисними для відповіді на питання про те, як створюються та виконуються завдання в процесі розробки.

У попередніх розділах було введено поділ завдань, що виконуються в програмному процесі, на завдання зі станом, пов’язані з реалізацією нових завдань, і повторювані, роботи з якими виконуються періодично. Цей поділ може призвести до припущення, що завдання з підтримкою стану, пов’язані з новими функціями, в основному створюються на початку проекту. Що стосується помилок, то було б розумно припустити, що вони виникають після реалізації певних вимог або функцій. Сама назва повторюваних завдань (наприклад, щоденне вставання) говорить про те, що вони виконуються з однаковою інтенсивністю протягом усього виробничого процесу. Створення нових завдань дуже важливе при плануванні процесу розробки, оскільки реалізацію завдань не можна починати, коли вони ще не створені. Цей факт обмежує розмір запланованої команди проекту.

Модель типів завдань і дослідження, проведені на реальних проектах, показують, як цей випадок виглядає насправді. На рисунку 2.8 показано, коли були створені завдання зі збереженням стану в досліджуваних проектах. Діаграми показують не кількість створених завдань, а їх трудомісткість, що краще відображає загальний розмір завдань, створених за місяць. DEV – нові

функції; BUG-DEV – дефекти, виявлені тестувальниками; BUG-CLI – дефекти, виявлені замовником (докладніше див. у таблиці 2.2).

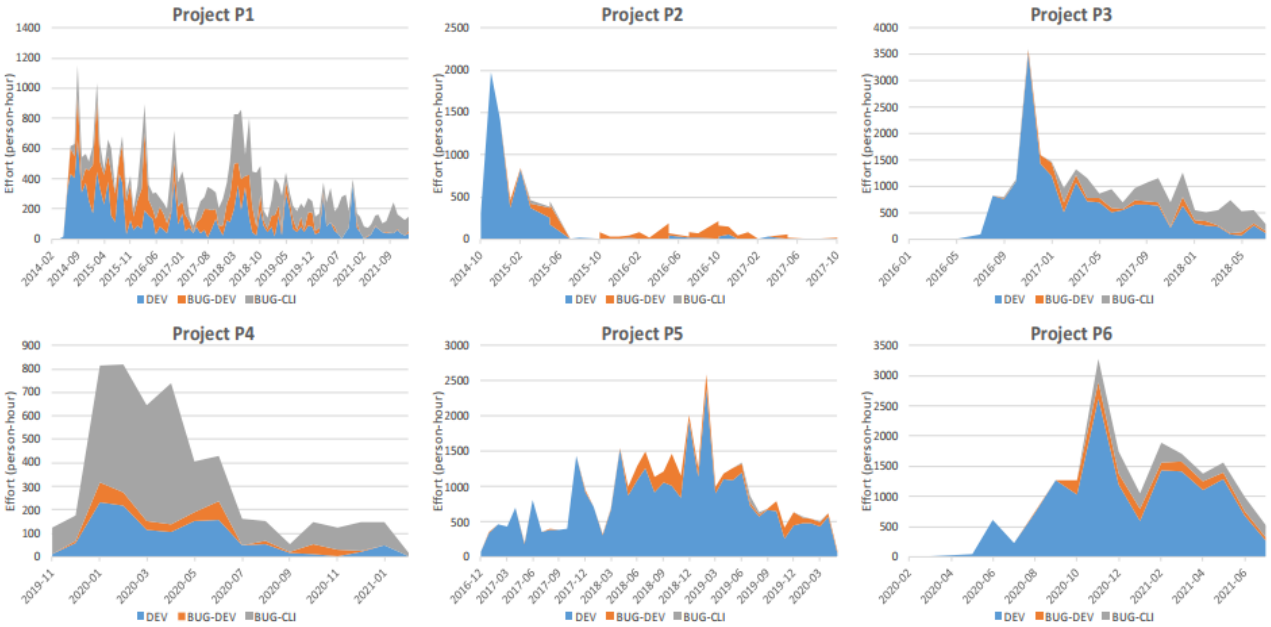

Рисунок 2.8 – Завдання зі збереженням стану, створені в процесі розробки досліджуваних проектів

Більшість діаграм показує, що нові функції розробляються протягом життя проекту. Це явище може стати великою несподіванкою, особливо тому, що конструкції P1-P4 були створені відповідно до гібридного підходу, в якому велика частка традиційних практик. Ситуація в довгостроковому проекті P1 зрозуміла, тому що він складається з багатьох етапів, і в кожній з них створювалися нові функції, які потрібно реалізувати. Проект P2 є винятком серед розглянутих проектів, тому що нові функції створюються протягом перших 8 місяців на початку процесу розробки, а потім, протягом 24 місяців, вони впроваджуються, і створюються виправлення дефектів, виявлених командою розробників.

Однак графіки проектів P3–P6 показують, що нові функції створюються до самого кінця виробничого процесу, хоча й у меншій мірі. Причини цього цікаві. Чи є це результатом тривалого процесу отримання нових вимог, паралельного процесу розробки? А може, причина в ознайомленні з деталями раніше

отриманих вимог? На жаль, дані, розміщені в системі Jira, не дають прямої відповіді на ці запитання, оскільки вони не враховують процеси розробки вимог. Цікавим явищем також є періодичне збільшення і зменшення як роботи над новими функціями, так і ремонту виявлених дефектів.

Робота над впровадженням державних завдань відбувається в іншому ритмі, ніж створення нових державних задач. Кількість людино-годин, що використовуються на місяць для виконання завдань із збереженням стану, залежить від кількості людей у команді проекту та їхніх призначених ролей. Слід враховувати, що команда також виконує повторювані завдання. На рисунку 2.9 наведено місячні витрати на виконання завдань із збереженням стану у досліджуваних проектах.

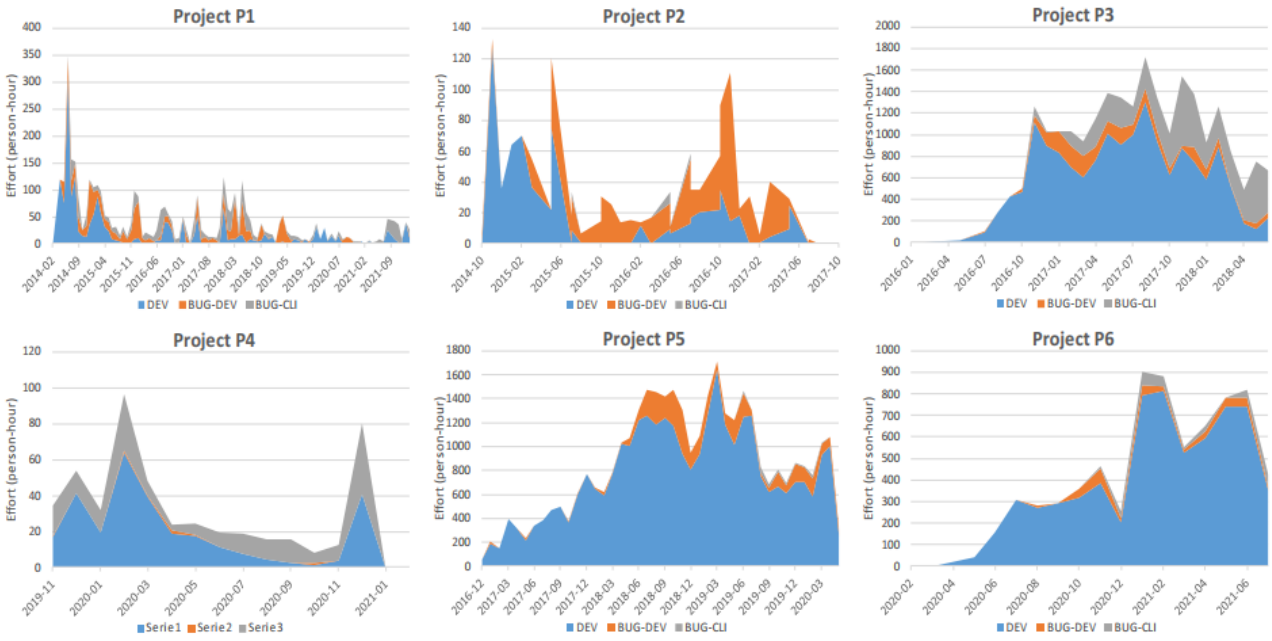

Рисунок 2.9 – Робота над завданнями із збереженням стану в досліджуваних проектах

На діаграмах видно, що в більшості проектів виправлення помилок, виявлені командою розробників, затримуються у зв’язку з реалізацією функцій розробленого програмного забезпечення. Ще більше затягується виправлення помилок, виявлених клієнтом, оскільки вони виявляються останніми. Це явище

найкраще видно на графіках проектів P3 і P6. Порівняння робочих графіків із графіками створених завдань (див. рис. 2.8) дає краще уявлення про проект. Наприклад, у проекті P2 після створення багатьох тисяч нових функцій робиться перерва на кілька місяців, а потім виникають дефекти, виявлені командою проекту. Графік роботи проекту P2 показує, що перерви в проекті не було, робота була менш інтенсивною, але на той момент виправлення дефектів, а потім впровадження нових функцій тривали .

Відповідь на запитання про те, як виглядає повторювана робота в процесі розробки програмного забезпечення, можна знайти на рисунку 2.10. Графіки показують подібну періодичність, як і графіки зусиль завдань із збереженням стану. Джерело цих періодичних збурень дуже цікаве. Ймовірно, певною мірою витрати на поточну та повторювану роботу є взаємодоповнюючими, тобто збільшення на першому графіку відповідає зменшенню на другому.

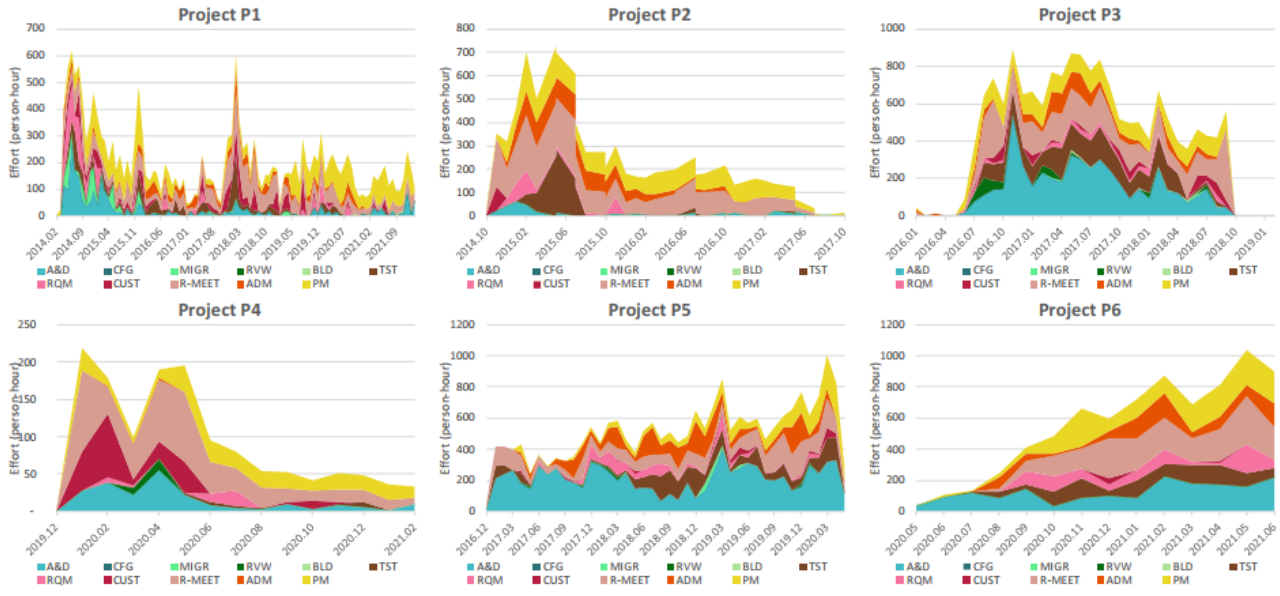

Рисунок 2.10 – Робота над повторюваними завданнями в досліджуваних проектах

Модель типів завдань у поєднанні з ролями членів команди проекту дозволяє аналізувати роботу, виконану в проекті. На цій основі можна простежити виконання завдань і відтворити склад команди проекту. Наступним

кроком у розробці стане можливість моделювання роботи в процесі розробки ПЗ або планування складу команди проекту на основі планів завдань.

Команда проекту складається з ролей і кількості робочих місць людей, зайнятих у певній ролі. Джерелом моделі є аналіз фактичних даних проекту, звідси відсутність певних ролей у команді, наприклад, ролі аналітика. Поточний набір ролей визначено наступним чином: ADM – адміністратор, PM – керівник групи, PRG – розробник (який також займається дизайном і збором вимог), TST – тестувальник. Команда складається з ролей і кількості позицій для даної ролі.

Прийнятий набір ролей не узгоджується з гнучким підходом, представленим методологією Scrum. Команда Scrum в основному складається з трьох ролей: Scrum master, product owner і команда розробників. Розробники – це всі члени команди розробників, які беруть участь у розробці програмного забезпечення. Однак на практиці варто розрізняти роль тестувальника (основною діяльністю якого є тестування програмного забезпечення та забезпечення якості), програміста (який створює робочий код) та адміністратора (який керує середовищем розробки та виробництва та інструментами підтримки розробки та впровадження нового програмного забезпечення).

На рисунку 2.11 показано концепцію зв’язку між типами завдань і ролями членів команди проекту. Він складається з типів завдань, ролей і двох видів зв'язків: простих і пропорційних. Простий зв’язок між типом завдання та роллю визначає, що завдання певного типу виконуються членами команди проекту з цією роллю. Наприклад, завдання DEV-PRG виконують люди з роллю PRG, а завдання BUG-CLI-TST – люди з роллю TST.

Існує пропорційний зв’язок між завданнями зборів і всіма ролями команди проекту. Ідея пропорційного підключення полягає в розподілі людино-годин, виділених на зустрічі, між людьми з команди проекту пропорційно кількості людей, які виконують певну роль. Наприклад, протягом місяця записується 100 людино-годин зустрічей, і програмісти виконали 72% роботи за місяць, тому 72

людино-години зустрічей на місяць додаються до робочого навантаження завдань розробників.

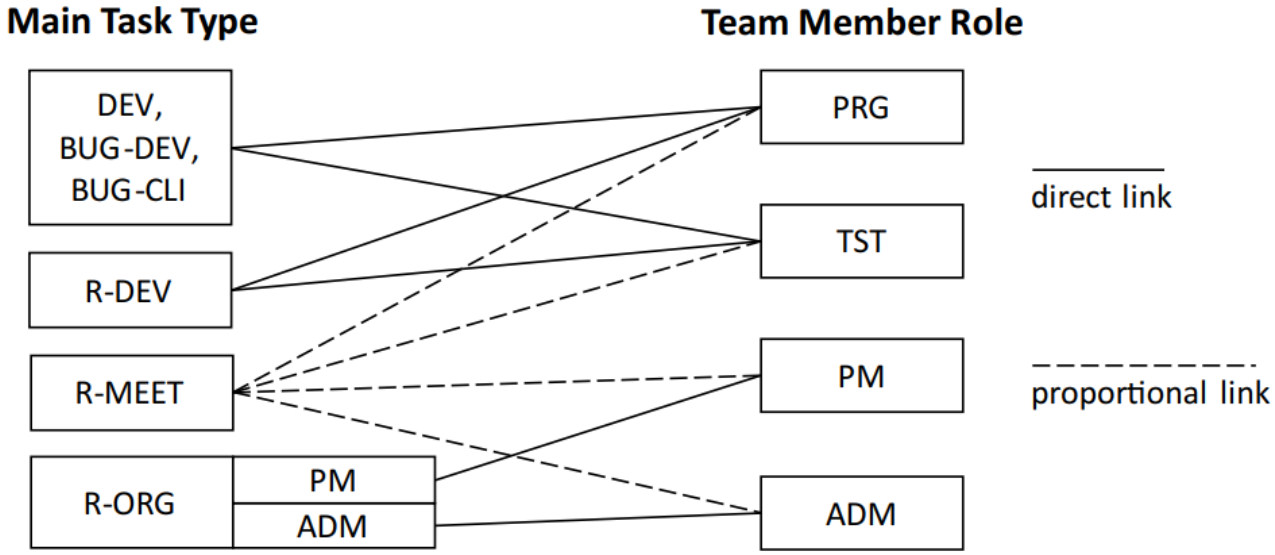

Рисунок 2.11 – Зв’язок між основними типами завдань і ролями членів команди проекту, які їх виконують

Робоче навантаження завдань, що виконуються щомісяця за ролями, ділиться на прийняту середню кількість годин, відпрацьованих особою за місяць. Оцінка – це кількість позицій для цієї ролі. Кількість позицій квантується до 1/4 і округлюється в більшу сторону. Прийняте середньооблікове число 165 годин роботи на місяць враховує тільки неробочі дні. Не враховує відпустки та можливі звільнення через хворобу працівника.

Пов’язування типів завдань із ролями членів команди проекту дає змогу приблизно визначити склад команди, необхідний для завершення проекту. У таблиці 2.3 представлено місячну роботу проекту P6 у розбивці за основними типами завдань і склад команди проекту, відбудованої на їх основі.

Таблиця 2.3 – Щомісячна робота проекту P6 та реконструйований склад команди проекту (від липня 2020 р.)

Місяць	DEV	Фактична щомісячне трудомісткість		R-Org	Кількість робочих місць у команді			
		R-DEV	P-Meet		ADM	PRG	TST	PM
01	155	-	-	11	0,25	1	0,25	0,25
02	307	3	-	8	0,25	1.75	0,25	0,25
03	284	41	21	106	0,5	2	0,5	0,25
04	290	113	65	105	0,5	2.5	0,5	0,5
05	361	191	133	128	0,25	3	1.25	1
06	467	200	129	293	0,25	3.5	1.25	2
07	255	130	256	158	0,5	2.75	0,5	1
08	901	212	202	292	1	6.25	1.5	1.25
09	881	193	208	328	1.25	6.25	1.5	1.25
10	555	138	152	293	0,75	3.25	1.75	1.5
11	650	151	198	333	0,75	4.25	1.5	1.75
12	782	269	289	326	0,5	6.5	1.25	2
13	821	110	196	404	1	5.75	0,75	1.75
14	423	54	114	184	0,25	2.75	0,75	1.25

Кількість людей у команді проекту зростає та зменшується відповідно до щомісячних поточних завдань. Команда зростала з початку проекту. У липні та жовтні 2020 року він був найвищим; кількість позицій становила 10,25. Після цього команда скоротилася, і проект завершився в грудні 2020 року. Була максимальна кількість програмістів 6,5 і 1,75 позицій тестувальників на проект. Цікаво, що у квітні та жовтні 2020 року в команді було дві посади керівника проекту. Часто проекти наймають людей для виконання організаційної та допоміжної роботи, наприклад, управління проблемами в системі Jira, обслуговування дощок Kanban або створення звітів.

2.3 Перевірка моделі

Алгоритм класифікації, згідно з рисунком 2.4, складається з методу поділу на підзавдання та призначення типів завдань на основі набору ключових слів, створених із ручної класифікації завдань проекту P3. Оскільки загальна кількість

проблем у всіх проектах була великою, класифікувати їх вручну було важко. Дослідження має служити доказом концепції, тому той самий набір ключових слів використовувався для класифікації завдань інших проектів.

Метод поділу завдань на підзадачі тісно пов'язаний з класифікацією завдань, оскільки типи підзадач впливають на відмінність, наприклад, від того, чи було виконано впровадження чи тестування. Якщо підзадача визначена неправильно, їй не буде присвоєно трудомісткість і вона не буде включена в показник точності EA_p. Розподіл підзавдань є складним, оскільки люди, які записують роботу в систему JIRA, роблять це різними способами. Метод поділу підзадач спочатку був розроблений для проекту P3, потім був адаптований до інших проектів. Основним показником ефективності класифікації проектних завдань EA_c – коефіцієнт трудомісткості розпізнаних підзадач (E^R_p) до загальної трудомісткості проекту E_p:

$$EA_p = \frac{E_p^R}{E_p} \cdot 100\% \tag{2.1}$$

де p – проект;

EA_p – індекс ефективності класифікації трудомісткості проектних завдань проекту p;

E^R_p – трудомісткість правильно визначених підзадач проекту p;

E_p – загальна трудомісткість проекту p.

Для подальшої перевірки правильності класифікації завдань проводиться ручна перевірка за зразком було проведено 100 випадково вибраних завдань. В результаті отримано показник точності (NA^M_p) визначення відсотка правильно класифікованих завдань у вибірці (N_p^{MC}) до кількості завдань у вибірці (N_p^M):

$$NA_p^M = \frac{N_p^{MC}}{N_p^M} \cdot 100\%, \tag{2.2}$$

де p – проект;

HA^{M}_{p} – показник ефективності класифікації кількості завдань проекту p у вибірці;

N_{p}^{MC} – кількість правильно визначених проектних завдань p у вибірці;

N_{p}^{M} – кількість проектних завдань p у вибірці,

$N_{p}^{M} = 100$.

Показник точності трудомісткості (EA^{M}_{p}) отримуємо, визначаючи відсоток трудомісткості правильно класифікованих завдань у вибірці (E_{p}^{MC}) до загальної трудомісткості вибірки (EA^{M}_{p}):

$$EA_{p}^{M} = \frac{E_{p}^{MC}}{E_{p}^{M}} \cdot 100\% \qquad (2.3)$$

де p – проект;

EA^{M}_{p} – індекс ефективності класифікації трудомісткості проектних завдань p у вибірці;

E_{p}^{MC} – трудомісткість правильно визначених проектних завдань p у вибірці;

E_{p}^{M} – загальна трудомісткість проекту p у вибірці.

У таблиці 2,4 наведено результати перевірки ефективності класифікації проектних завдань і підзадач та ручної перевірки зразків проектних завдань.

Таблиця 2.4 – Результати перевірки ефективності класифікації завдань і підзадач проектів

Проект	***EA***	***HA*** ^{M}	***E. A.*** ^{M}
P1	91,40%	93,10%	93,83%
P2	61,60%	67,20%	31,20%
P3	99,60%	98,20%	99,74%
P4	25,10%	24,20%	52,91%
P5	99,30%	87,10%	100,00%
P6	54,60%	68,30%	99,30%

Показники точності проекту P3 можна вважати зразковими, оскільки для цього проекту розроблено алгоритм класифікації. Відмінності значень числових і трудомістких показників точності, визначених при ручній перевірці малих вибірок (від 1% до 4% від кількості завдань у проекті), пов'язані з тим, що не всі правильно класифіковані завдання мають трудомісткість. годин, записаних. Низькі значення показників для проектів P2, P4 та P6 пов’язані з частим використанням іншої мови в описах завдань. Набір ключових слів, розроблений для проекту P3, складається з англійських слів, тому алгоритм класифікації погано переноситься на деякі проекти. Крім того, ручна перевірка проекту P4 виявила: відсутність ключових слів в описі та велику кількість завдань без зміни статусу, що призвело до відсутності підзавдань і завдань без зареєстрованих робіт. Результати в таблиці 4 вказують на необхідність перекладу посадових інструкцій із системи JIRA на англійську мову перед початком класифікації на основі ключових слів або використання моделей NLP.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

3.1 Поняття та об’єкт аналізу технічної безпеки

Безпеку визначають як стан діяльності людини, за яким з визначеною ймовірністю виключено прояв небезпек або ж відсутня надзвичай-на небезпека. Безпека праці – це стан умов праці людини, за яких відсутня дія небезпечних і шкідливих факторів.

Об’єктом аналізу безпеки праці є виробнича система "людина – машина – навколишнє середовище" (ЛМС), в якій в єдиній комплекс, створений для виконання певних функцій, поєднані технічні об’єкти, люди і навколишнє середовище, які взаємодіють між собою.

Основними компонентами виробничої системи є людина, машина, навколишнє середовище, взаємодія між якими має ґрунтуватись на дотри-манні відповідних правил, нормативних документів і бути керованою.

Система ЛМС є багаторівневою за ієрархією управління. Ієрархія поділяє людей на особу, яка формує завдання, організовує й управляє виробництвом, й особу, яка разом з технікою безпосередньо виконує це завдання. Таким чином, людина системи ЛМС більш високого рівня розглядає людину і техніку системи ЛМС більш низького рівня як єдиний компонент – своєрідну людину-машину, призначену для здійснення замислу.

Крім рівнів і компонентів в системі ЛМС доцільно виділити окремі стадії її життєвого циклу:

1. Стадія проектування (визначення завдань, формування вимог, розрахунок параметрів).

2. Стадія реалізації (коли у процесі виробництва перша стадія реалізується на практиці).

3. Стадія експлуатації (коли система ЛМС здійснює покладені на неї робочі функції).

Вірогідність нещасного випадку зростає, як тільки людина попадає в поле дії небезпечного або шкідливого фактору. Це небезпечні зони, що характеризуються певним видом небезпеки, її інтенсивністю, часом і простором дії.

Таким чином, з точки зору аналізу й управління небезпеками необхідно розглядати та аналізувати структурні елементи системи ЛМС – рівні (вищий і нижчий), компоненти і стадії життєвого циклу.

Взаємодія компонентів, що входять до системи ЛМС, може бути штатною і нештатною. Нештатна взаємодія може виявлятися у вигляді надзвичайної події – небажаних, незапланованих випадків, що порушують технологічний процес у відносно короткий відрізок часу. Відмова й інцидент, як правило, передують надзвичайній події, але можуть мати і самостійне значення. До головних моментів аналізу небезпек належить пошук відповідей на такі питання:

1. Які об’єкти є небезпечними;
2. Яким надзвичайним подіям можна запобігти;
3. Які надзвичайні події неможливо усунути і як часто вони мати-муть місце;
4. Яку шкоду не усунуті надзвичайні події можуть спричинити людям, об’єктам, навколишньому середовищу.

Пошук причин надзвичайних подій призводить до аналізу системи управління небезпеками (СУН) на виробництві. Ці системи обов’язково включають такі компоненти, як наявність інформації, зворотних зв’язків та алгоритми функціонування.

Наявність зворотних зв’язків й інформаційної системи дозволяє проводити збір даних щодо відхилень, відмов, проводити аналіз небезпек, порівнювати наслідки функціонування системи ЛМС з програмою управління небезпеками, приймати рішення. У виробничій системі ЛМС інформаційні

функції виконують: рапорти інспекторів, акти розслідування нещасних випадків, аварій, протоколи атестації робочих місць тощо.

3.2 Розрахунок захисного заземлення

Захисне заземлення повинне забезпечити захист людей від ураження електричним струмом, при дотику до металевих частин, які можуть виявитися під напругою. Заземленням називається навмисне з'єднання електроустановок із заземлюючим пристроєм. Заземлювачем називається провідник, що перебуває в контакті із землею або її еквівалентом. Заземлюючим провідником називається провідник, що з'єднує заземлені частини із заземлювачем. Сукупність з'єднуючих провідників і заземлювачів називається заземлюючим пристроєм. Для установок потужністю не більше 100 кВт опір заземлюючого пристрою не повинне перевищувати 10 Ом, для установок потужністю більше 100 кВт – 4 Ом.

Розрахунок штучного заземлювального пристрою при відсутності природних заземлювачів.

Вихідні дані:

Захищуваний об’єкт – комп’ютерна мережа.

Захищуваний об’єкт – стаціонарний.

Напруга мережі – 220 В.

Виконання мережі – з глухозаземленою нейтраллю.

Тип заземлювального пристрою – вертикальні труби.

Розміри вертикальних заземлювачів:

Довжина – 6 м.

Діаметр труби – 0,60 м.

Товщина стінки труби – 0,06 м.

Висота труби – 0,6 м.

Відношення відстані між трубами до їхньої довжини:

$$\frac{L_B}{l_B} = 1 \qquad (3.1)$$

Розмір горизонтального заземлювача (з'єднувальної стрічки): довжина $L_Г = L_{З.С.}$ $L_Г = L_{З.С.}$– згідно з розрахунком, м; ширина горизонтальної з'єднувальної стрічки b_c=0,04 м.

Глибина закладання вертикальних заземлювачів h_B=0,6.

Розміщення заземлювачів попередньо приймають за чотирикутним контуром при числі стержнів від 4 до 100 та в один ряд при числі стержнів від 2 до 20.

Грунт – супісок; склад – однорідний; вологість – мала; агресивність – нормальна.

Кліматична зона – II.

Розрахунок:

1. Визначаємо характеристику навколишнього середовища в приміщенні організації: за пожежною небезпекою згідно з ПУЕ воно відноситься до класу П-II; за вибухонебезпекою згідно з ПУЕ – до класу В-I; за ступенем ураження електричним струмом – без підвищеної та особливої небезпеки.

2. Визначаємо $R_Д$ – допустиме (нормативне) значення опору розтікання струму в заземлювальному пристрої, $R_Д \leq 4$ *Ом*.

3. Обраховуємо $K_{С.В.}$ $K_{С.В.}$ – приблизне значення питомого опору ґрунту, що рекомендується для розрахунку – $\rho_{ТАБЛ} = 300$ *Ом* · *м* $\rho_{ТАБЛ} = 3000$ м*м.

4. Визначаємо $K_{С.В.}$ – коефіцієнт сезонності для вертикальних заземлювачів для денної кліматичної зони. За довідковою інформацією приймаємо $K_{С.В.}$=1,5.

5. Обраховуємо значення $К_{С.Г.}$ – коефіцієнт сезонності для горизонтального заземлювача згідно з кліматичною зоною. За довідковою інформацією приймаємо$К_{С.Г.} = 3,5$;$К_{С.Г.}$=3,5.

6. Визначаємо $\rho_{РОЗР.В.}$ $\rho_{РОЗР.В.}$ – розрахунковий питомий опір ґрунту для вертикальних заземлювачів:

$$\rho_{РОЗР.В.} = \rho_{ТАБЛ} \cdot К_{С.В.} = 300 \cdot 1,5 = 450\ Ом \cdot м. \quad (3.2)$$

$\rho_{РОЗР.В.} = \rho_{ТАБЛ}$*$К_{С.В.} = 300$*$1,5 = 450$Ом*м.

7. Розраховуємо $\rho_{РОЗР.Г}$ – розрахунковий питомий опір ґрунту для горизонтальних заземлювачів:

$$\rho_{РОЗР.Г} = \rho_{ТАБЛ.} * К_{С.Г.} = 300 * 3,5 = 1050Ом * м.$$

$$\rho_{РОЗР.Г.} = \rho_{ТАБЛ} \cdot К_{С.Г.} = 300 \cdot 3,5 = 1050\ Ом \cdot м. \quad (3.3)$$

8. Обраховуємо t – відстань від поверхні землі до середини вертикального заземлювача:

$$t = h_B + \frac{l_B}{2} = 0,6 + \frac{6}{2} = 3,6\ м. \quad (3.4)$$

9. Визначаємо R_B – опір, Ом, розтікання струму в одному вертикальному заземлювачі:

$$R_B = \frac{\rho_{РОЗ.В}}{2\pi l_B} \cdot \left(ln\frac{2l_B}{d} + 0,5 \cdot ln\frac{4t + l_B}{4t - l_B} \right) =$$
$$= \frac{450}{2\pi \cdot 6} \cdot \left(ln\frac{2 \cdot 6}{0,60} + 0,5 \cdot ln\frac{4 \cdot 3,6 + 6}{4 \cdot 3,6 - 6} \right) = 41,05\ Ом \quad (3.5)$$

10. Визначаємо $n_{т.в.}$ – теоретична кількість вертикальних заземлювачів без врахування коефіцієнта використання $\eta_{в.в.}$, тобто $\eta_{в.в.}=1$:

$$n_{Т.В.} = \frac{R_В}{R_Д \cdot \eta_{В.В}} = \frac{41{,}05}{4 \cdot 1} = 10{,}26 \; шт. \qquad (3.6)$$

11. Визначаємо $\eta_{в.в}$ – коефіцієнт використання вертикальних заземлювачів при розташуванні їх згідно з вихідними даними або за чотирикутним контуром при числі заземлення, $n_{т.в.}=11$ та при відношенні $\frac{L_В}{l_В} = 1$. За довідником приймаємо $\eta_{в.в}$ =0,42.

12. Визначаємо $n_{н.в}$ – необхідна кількість штук, вертикально однакових заземлювачів з врахування коефіцієнта використання:

$$n_{Н.В} = \frac{R_В}{R_Д \cdot \eta_{В.В}} = \frac{41{,}05}{4 \cdot 0{,}42} = \frac{41{,}05}{1{,}68} = 24{,}43 \; шт. \qquad (3.7)$$

13. Визначаємо $R_{розр.в}$ – вертикальний опір, Ом, розтіканню струму у вертикальному заземленні при $n_{н.в} = 24{,}43$ без врахування з'єднувальної стрічки:

$$R_{РОЗР.В} = \frac{R_В}{n_{Н.В} \cdot \eta_{В.В}} = \frac{41{,}05}{10{,}26} = 4 \; Ом. \qquad (3.8)$$

14. Визначаємо $L_в$ – відстань між вертикальним заземлювачами за відношенням $\frac{L_В}{l_В} = 1$, звідси

$$L_В = 1 \cdot l_В = 1 \cdot 6 = 6 \; м. \qquad (3.9)$$

15. Визначаємо $L_{з.с}$ – довжину, м, з'єднання стрічки горизонтального заземлювача:

$$L_{З.С} = 1{,}05 \cdot L_B(n_{Н.В} - 1) = 1{,}05 \cdot 6(24{,}43 - 1) = 147{,}60\ м. \quad (3.10)$$

16. Визначаємо $R_{г.з.с}$ – опір, Ом, розтікання струму в горизонтальному заземлювачі (з'єднувальній стрічці):

$$R_{Г.З.С} = \frac{\rho_{РОЗ.Г}}{2 \cdot \pi \cdot L_{З.С}} \cdot ln\frac{2 \cdot L_{З.С}^{\ 2}}{2 \cdot b \cdot t} = \frac{1050}{2 \cdot 3{,}14 \cdot 147{,}61} \cdot ln\frac{2 \cdot (147{,}61)^2}{2 \cdot 0{,}04 \cdot 3{,}6} = 13{,}50\ Ом. \quad (3.11)$$

17. Визначаємо $\eta_{в.г}$ – коефіцієнт використання горизонтального заземлення при розташуванні вертикальних заземлювачів згідно з вихідними даними або за чотирикутним контуром при відношенні $\frac{L_B}{l_B} = 1$ та необхідній кількості вертикальних заземлювачів $n_{н.в}$ = 24,43.

За довжину приймаю $\eta_{в.г}$ = 0,19.

18. Визначаємо $R_{розр.г}$ – розрахунковий опір, Ом, розтікання струму в горизонтальному заземленні (з'єднувальній стрічці) при числі електродів $n_г$= 1:

$$R_{РОЗР.Г} = \frac{R_{Г.З.С}}{n_Г \cdot \eta_{В.Г}} = \frac{13{,}5}{1 \cdot 0{,}19} = 71{,}05\ Ом. \quad (3.12)$$

19. Визначаємо $R_{розр.в.г}$ – розрахунковий теоретичний опір, Ом, розтікання струму у вертикальному та горизонтальному заземленні:

$$R_{РОЗР.В.Г} = \frac{1}{\frac{1}{R_{РОЗ.В}} + \frac{1}{R_{РОЗ.Г}}} = \frac{1}{\frac{1}{4} + \frac{1}{71{,}05}} = 3{,}78\ Ом. \quad (3.13)$$

20. Вибираємо матеріал та поперечний перетин з'єднувальних провідників. За довідковою інформацією вибираю голі мідні $S_м$ = 4 мм^2 провідники.

21. Вибираємо матеріал та поперечний перетин магістральної шини. За довідковою інформацією обираємо сталеву шину товщиною $\delta_с$ = 4мм і перетином не менше δ = 100 мм^2.

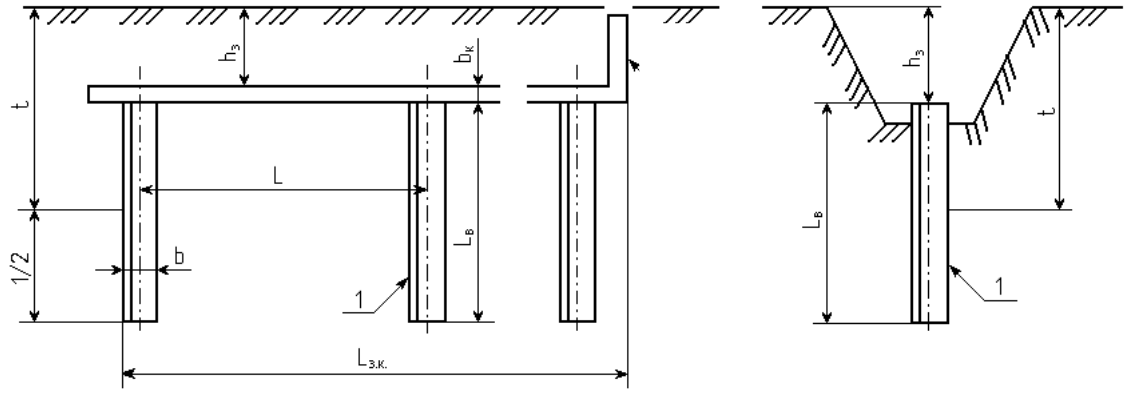

Рисунок 3.1 – Схема заземлювального контуру:

1 – вертикальний заземлювач; 2 – горизонтальний заземлювач;

$h_З$ – глибина закладання заземлювачів; L – відстань між заземлювачами;

$b_К$ – ширина квадрата; t – відстань від середини заземлювача до поверхні грунту; $L_{З,К}$ довжина горизонтального заземлювача; d – ширина кутника;

$L_В$ – довжина вертикального заземлювача.

Схема з'єднання обладнання з магістральною шиною та з'єднання магістральної шини з заземлювальним пристроєм (рисунок 3.1).

ВИСНОВОК

Дані реальних проектів є основою представленого дослідження та моделі. З одного боку, вони збільшують можливості практичного застосування, з іншого боку, вони обмежують модель до типів завдань і ролей, присутніх у досліджуваних проектах. Зважаючи на це, ми намагалися зробити модель гнучкою та відкритою для модифікацій.

Підключення моделі до системи Jira дозволяє легко отримувати дані для аналізу та збільшує її комерційний потенціал. Окремий абстрактний рівень моделі в поєднанні з виділеною базою даних підтримує можливість створення інтерфейсів для інших систем управління IT-проектами.

Наведені в роботі алгоритми класифікації базуються на ручному розпізнаванні типів завдань. При розпізнаванні вручну створюються правила на основі ключових слів, що дозволяє автоматично розпізнавати типи завдань при наступних повтореннях. Як відомо з літератури та практики, такі алгоритми не дуже еластичні та не надто точні (45% точності) [19]. Однак із зростаючою базою завдань, розпізнаних вручну, буде легко перейти до моделей NLP. Це дозволить повністю автоматизувати роботу алгоритму класифікації завдань і підзадач у режимі реального часу. Це дозволить аналізувати дані, зібрані в JIRA, створювати звіти та діаграми, щоб забезпечити уявлення про виробничий процес, і підтримувати керівника проекту в прийнятті рішень.

Розподіл завдань на стан і циклічні показує, що завдання стану створюються в процесі розробки програмного забезпечення, а їх кількість і трудомісткість залежать від розміру проекту. Темпи зростання та виконання державних завдань залежать від складу команди проекту. Від цього темпу залежить термін виконання проекту. Кількість і трудомісткість циклічних завдань, навпаки, залежить від тривалості проекту. Таким чином, класифікація завдань стає основою для побудови генератора станових і циклічних завдань для створення планів розробки ПЗ. У свою чергу, створення плану розробки та

складу команди проекту дозволить побудувати спрощену симуляцію роботи в проекті.

Можливість створити план проекту та підібрати відповідний склад команди проекту, а потім, завдяки симуляції, перевірити, як буде проходити робота, дозволить комплексно підтримувати управління процесом розробки. Завдяки моделюванню можна буде оцінити, чи відповідає склад команди специфіці проекту. Моделювання може показати, що, наприклад, програмісти реалізували вимоги і що команда чекає, поки тестувальники виконають тести. Таким чином керівник проекту може розпізнати ризик вузького місця в проекті та запобігти йому заздалегідь.

Планування проекту корисно не тільки перед початком. Автоматична класифікація завдань у режимі реального часу дозволить аналізувати та використовуватиме розраховану статистику завдань для планування наступних спринтів або етапів процесу розробки з підвищенням точності.

Введення додаткової класифікації завдань з точки зору бізнес- і технологічних компонентів у поєднанні з моделлю компетенцій співробітників дозволить автоматично збирати інформацію про досвід співробітників у бізнес- і технологічних сферах. Це дозволить оцінити рівень компетенції співробітників, вибрати склад команди проекту і, можливо, керувати командою таким чином, щоб компетенції дублювалися та розпорошувалися між членами команди.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wysocki, J. The use of information technologies in the enterprise. Science about the Enterprise: Selected Issues; Lichniak I.: Warsaw, Poland, 347-377.
2. Волович, В., Береженко, Б. М., & Боднарчук, І. О. (2022). Задача проєктування програмної архітектури в процесах забезпечення якості. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 104-106.
3. Гузеляк, О., Шевчук, Ю., Береженко, Б. М., & Боднарчук, І. О. (2022). Програмна архітектура в розподілених командах гнучких проєктів. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 110-112.
4. Боднарчук, І., Харченко, О., Хоміцький, Б., & Шимчук, Г. (2019). Проектування архітектури програмних систем в проектах з гнучкими методами управління. Матеріали XXI наукової конференції Тернопільського національного технічного університету імені Івана Пулюя, 46-48.
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. Kędziora, A. F. SCRUM Methodology in Small and Medium IT Projects. 2011.
7. 10. Shore, J.; Warden, S. The Art of Agile Development, 2nd ed.; O’Reilly Media, Inc.: Sebastopol, CA, USA, 2008.
8. 11. Bielec, J. The same technology—Different results. Key elements of the success of an IT project implementation. In Proceedings of the 13th PLOUG Kos´cielisko Conference, Post-Conference Materials, Kos´cielisko, Poland, 17–20 October 2007.

9. Wróbleski, P. IT Project Management for Practitioners; Helion Publishing House: Warsaw, Poland, 2005.
10. 20. Kruchten, P. The Rational Unified Process: An Introduction; Addison-Wesley Professional: Boston, MA, USA, 2004.
11. Manifest Agile. 2001. Available online: https://agilemanifesto.org/iso/pl/principles.html (accessed on 17 June 2022).
12. Łabuda, W. How to implement a successful IT project. In Proceedings of the Conference Materials Summarizing the Project Program for the Development of the Teaching Offer and Improving the Competences of Lecturers at the Warsaw University of Information Technology, Warsaw, Poland, 20–21 September 2011.
13. Raunak, M.S.; Binkley, D. Agile and Other Trends in Software Engineering. In Proceedings of the IEEE 28th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 25–28 September 2017.
14. Highsmith, J. APM: Agile Project Management. How to Create Innovative Products; Mikom: Warsaw, Poland, 2005.
15. Akbar, M.A.; Smolander, K.; Mahmood, S.; Alsanad, A. Toward successful DevSecOps in software development organizations: A decision-making framework. Inf. Softw. Technol. 2022, 147, 106894.
16. Anwar, A. (2014). A review of rup (rational unified process). International Journal of Software Engineering (IJSE), 5(2), 12-19.
17. West, D. Water-Scrum-Fall Is the Reality of Agile for Most Organizations Today; Forrester Research: Cambridge, MA, USA, 2011; Volume 26. 50. Attri, R.; Grover, S.; Dev, N.; Kumar, D. Analysis of barriers of total productive maintenance (TPM). Int. J. Syst. Assur. Eng. Manag. 2013, 4, 365–377.
18. Akbar, M.A.; Sang, J.; Nasrullah; Khan, A.A.; Mahmood, S.; Qadri, S.F.; Hu, H.; Xiang, H. Success factors influencing requirements change management process in global software development. J. Comput. Lang. 2019, 51, 112–130.
19. Mccallum, A.; Kamal, N. Text Classification by Bootstrapping with Keywords, EM and Shrinkage; ACL: Stroudsburg, PA, USA, 2001.

20.Жидецький, В. Ц., Джигирей, В. С., & Мельников, О. В. (2000). Основи охорони праці. Львів: Афіша, 350, 132-136.

21.Навакатіян О.О., Кальниш В.В., Стрюков С.М. Охорона праці користувачів комп’ютерних відеодисплейних терміналів. - К.:1997. - 400с.