**Державна служба України з надзвичайних ситуацій**

**Львівський державний університет безпеки життєдіяльності**

**Юлія НАЗАР, Ольга СМОТР, Сергій БУРЛАКОВ**

# МОДЕЛІ ТА ЗАСОБИ АДАПТИВНОГО ПЛАНУВАННЯ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БЕЗПЕКО-ОРІЄНТОВАНОГО СПРЯМУВАННЯ

**Монографія**

Львів – 2025

УДК 004.4:004.5

**Ю.С. Назар, О.О. Смотр, С. В. Бурлаков. Моделі та засоби адаптивного планування життєвого циклу програмного забезпечення безпеко-орієнтованого спрямування.** – Львів: ЛДУ БЖД, 2025. – 95с.

**Рецензенти:**

***Зачко Олег**, доктор технічних наук, професор, заслужений діяч науки і техніки, професор кафедри права та менеджменту Львівського державного університету безпеки життєдіяльності*

***Рак Тарас,** доктор технічних наук, професор, професор приватного закладу вищої освіти «ІТ СТЕП Університет»*

***Павлова Ольга,** доктор філософії, доцент, завідувач кафедри комп'ютерної інженерії та інформаційних систем Хмельницького національного університету*

Монографія орієнтована забезпечення освітньо-наукового процесу здобувачів освіти третього освітньо-наукового рівня зі спеціальностей 122 «Комп’ютерні науки» присвячена підвищенню ефективності створення спеціалізованого програмного забезпечення шляхом розроблення нових моделей та засобів адаптивного планування життєвого циклу в динамічних умовах функціонування оперативних служб (зокрема ДСНС). У роботі представлено концептуальні засади управління розробленням безпеко-орієнтованих сервісів, що поєднують принципи гнучких методологій (Agile) зі специфікою роботи рятувальних підрозділів. Для реалізації поставлених завдань використано комплексний методологічний апарат, який включає: методи інженерії програмного забезпечення, понятійний апарат теорії множин для моделювання етапів життєвого циклу, теорію графів та мережі Петрі для імітаційного моделювання процесів планування та розрахунку часових параметрів. В монографії отримано такі основні результати: розроблено концептуальну модель управління життєвим циклом безпеко-орієнтованого програмного забезпечення в динамічних умовах; побудовано імітаційні моделі обходу мережевого графа із застосуванням мереж Петрі для автоматизації розрахунку параметрів планування; створено інформаційну технологію підтримки прийняття рішень, яка дозволяє здійснювати оперативне перепланування проєктних робіт; здійснено практичну апробацію результатів під час розробки безпеко-орієнтованих сервісів для Державної служби України з надзвичайних ситуацій.

Рекомендовано Вченою радою

Львівського державного університету безпеки життєдіяльності

(*Протокол №__ від ___ _____ 20__ року*).

ЗМІСТ

# ВСТУП

Існуючі моделі, методи та механізми гнучкого управління ІТ-проєктами, зокрема із розробки програмного забезпечення (далі – ПЗ), передбачають широкий спектр інструментальних засобів реалізації цілей проєкту. Відомі підходи апробовані реалізацією низки успішних проєктів у сфері аутсорсингу та продуктових компаній. Проте, як показує аналіз ринку праці та попереднє вивчення бізнес процесів у проєктах розробки спеціалізованого ПЗ, реалізація окремих етапів життєвого циклу розроблення такого ПЗ потребує залучення додаткових фахівців вузької галузі.

Особливої уваги у подібного роду проєктах потребує процес короткотермінового планування (планування спринтів). Проблема короткотермінового планування полягає у необхідності коректного визначення обсягу робіт та застосовуваних технологій, що, своєю чергою, впливає на успішність виконання робіт в умовах обмежених ресурсів. Зважаючи на відомі методи короткотермінового планування, можна зробити висновок про їх часткову невідповідність процесам планування обсягів робіт у проєктах розробки саме спеціалізованого ПЗ (далі – СПЗ), зокрема у галузі безпеки людини. Подібні проєкти, як правило, характеризуються значною невизначеністю вхідних даних та динамічними специфікаціями. Такі передумови створюють несприятливий клімат у команді розробників ПЗ, внаслідок чого дестабілізації зазнає продуктивність цієї команди.

Варто також зауважити, що членами команд розробки безпеко-орієнтованих сервісів, як правило, є кадрові працівники відповідних служб, які в міру своєї службової підготовки повинні поєднувати роботу, пов'язану із розробкою зазначених сервісів, з іншими різновидами оперативної та/або службової діяльності. Тому, зважаючи на специфіку роботи і особливих учасників проєктних команд із розробки БОС, динаміка проєктного середовища набуває дещо іншого значення.

Саме тому існує потреба у детальному аналізі та побудові нових методів, моделей та інформаційної технології для адаптивного планування розробки

безпеко-орієнтованих сервісів, що відповідатимуть парадигмі гнучкого управління процесом розробки програмного забезпечення в динамічних умовах оперативних формувань.

**Мета досліджень, представлених у монографії:** підвищення ефективності створення спеціалізованого програмного забезпечення шляхом розроблення нових моделей та засобів адаптивного планування життєвого циклу програмного забезпечення в динамічних умовах.

Для досягнення мети окреслено вирішення таких **завдань**:

- здійснити аналіз сучасного стану предметної області та наукових джерел щодо проблем підвищення ефективності управління життєвим циклом спеціалізованого програмного забезпечення;
- дослідити структуру та обсяги робіт на всіх етапах життєвого циклу програмного забезпечення, визначити їх взаємозалежності та обґрунтувати фундаментальні принципи автоматизації підтримки прийняття управлінських рішень при створенні програмних систем;
- побудувати концептуальну модель управління життєвим циклом розроблення спеціалізованого програмного забезпечення безпеко-орієнтованого спрямування;
- розробити імітаційні моделі аналізу мережевих графів планування для обчислення часових параметрів (ранніх та пізніх термінів, часових резервів) виконання завдань;
- створити алгоритми побудови та обробки мережевих моделей, спрямовані на підвищення ефективності планування окремих етапів створення програмного продукту;
- розробити компоненти засобів підтримки прийняття рішень для ітераційного планування, що базуються на моделях оцінювання спроможності команди розробників виконувати запланований обсяг робіт;
- виконати проєктування, програмну реалізацію та апробацію розробленої технології підтримки прийняття рішень у процесі створення безпеко-орієнтованих сервісів для потреб цивільного захисту.

**Об’єкт досліджень:** процеси опрацювання потоків інформації щодо управління життєвим циклом розроблення програмного забезпечення в динамічному оточенні.

**Предмет досліджень:** моделі, методи та засоби інформаційної технології оперативного прийняття рішень щодо адаптивного планування розробки спеціалізованого програмного забезпечення в динамічному оточенні.

**Методи дослідження:** теоретичні дослідження, методи математичного моделювання та методи інженерії програмного забезпечення. Теоретичні дослідження включають огляд літературних джерел, аналіз проблеми та теоретичне обґрунтування шляхів її вирішення. На цьому етапі проведено збір та аналіз статистичних даних щодо виконання проєктів зі створення безпеко-орієнтованих сервісів, описано процес розробки спеціалізованих програмних систем безпеко-орієнтованого спрямування із використанням теорії множин, а також узагальнено практичний досвід впровадження програмних систем у Державній службі України із надзвичайних ситуацій. На етапі математичного моделювання розроблено імітаційні моделі обходу мережевого графа із використанням мереж Петрі для розробки алгоритмів та автоматизації розрахунку параметрів мережевого графа планування ІТ-проєкту. Методи інженерії програмного забезпечення включали у себе розробку інформаційної технології підтримки прийняття рішень щодо управління життєвим циклом розроблення спеціалізованого програмного забезпечення, яка базується на обґрунтованих методах, моделях та алгоритмах для підвищення ефективності менеджменту життєвого циклу розроблення програмного забезпечення безпеко-орієнтованого спрямування.

**За результатами досліджень в монографії отримано такі результати:**

вперше розроблено:

- імітаційну модель обходу мережевого графа шляхом відтворення паралельних (розподілених) процесів мережами Петрі, що дозволяє її застосування для розробки алгоритмів та автоматизації розрахунку параметрів мережевого графа планування ІТ-проєкту;

- концептуальну модель процесу управління життєвим циклом розроблення програмного забезпечення, що корелює із принципами гнучкої методології управління ІТ-проєктами для підвищення якості менеджменту розроблення програмного забезпечення безпеко-орієнтованого спрямування в динамічних умовах.

удосконалено:

- метод планування життєвого циклу розроблення програмного забезпечення шляхом використання мережевого графа для можливості оперативного перепланування проєкту в динамічних умовах;

набула подальшого розвитку:

- інформаційна технологія, покладена в основу розроблення інформаційної системи підтримки прийняття рішень щодо управління життєвим циклом розроблення спеціалізованого програмного забезпечення, яка базується на обґрунтованих методах, моделях та алгоритмах для підвищення ефективності менеджменту життєвого циклу розроблення програмного забезпечення безпеко-орієнтованого спрямування.

**Теоретичне значення результатів роботи полягає у**:

- розробці інструментарію інформаційної системи прийняття рішень в управлінні життєвим циклом спеціалізованого програмного забезпечення;
- розробці імітаційної моделі обходу мережевого графа для визначення його основних показників;
- визначені теоретичних передумов використання існуючих моделей, методів та інформаційних технологій, що лежать в основі запропонованої системи підтримки прийняття рішень, яка враховує специфіку предметної галузі та оперативного менеджменту щодо управління життєвим циклом спеціалізованого програмного забезпечення.

**Практичне значення результатів полягає у** імплементації розроблених та науково-обґрунтованих методів, моделей і алгоритмів в процеси управління життєвим циклом розроблення спеціалізованого програмного забезпечення в динамічних умовах для Державної служби України з надзвичайних ситуацій.

# РОЗДІЛ І. КОНЦЕПТУАЛЬНІ ЗАСАДИ РОЗРОБЛЕННЯ СПЕЦІАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1. Аналіз традиційних та гнучких (Agile) методологій: переваги та обмеження для оперативних формувань

У контексті стрімкого розвитку інформаційних технологій та зростання потреби у високоспеціалізованих рішеннях, особливо для критично важливих сфер, як-от оперативні формування та служби реагування (наприклад, ДСНС), питання вибору ефективної методології управління розробкою програмного забезпечення набуває стратегічного значення. Спеціалізоване програмне забезпечення (СПЗ), а тим більше його підвид – безпеко-орієнтоване ПЗ (БОС), має унікальні вимоги, пов'язані з динамічністю середовища, критичністю даних, а також специфікою команд розробки, що часто складаються з кадрових працівників, які поєднують розробку з основною службовою діяльністю.

Традиційно в управлінні проєктами розробки виділяють два кардинально різні підходи: жорстко-послідовні, або традиційні (Waterfall), та адаптивні, або гнучкі (Agile) методології. Для звичайних ІТ-проєктів вибір залежить від багатьох факторів, проте для розробки БОС в умовах нестабільності та постійної зміни пріоритетів, характерних для оперативних формувань, необхідний ретельний аналіз переваг та обмежень кожного підходу. Цей підрозділ присвячений критичному огляду традиційних та гнучких методологій, щоб визначити їхню придатність та потенційні виклики у сфері розробки спеціалізованого програмного забезпечення.

Waterfall відноситься до традиційних методологій управління проєктами, яким притаманний чіткий та послідовний процес розробки програмного забезпечення. Не зважаючи на те, що на сьогодні дана модель майже не застосовується у сфері реалізації ІТ-проектів, вона є дуже важливою, адже на ній базуються усі інші методології розробки програмного забезпечення.

Традиційна методологія поділяє життєвий цикл на деякий набір фаз, кожну з яких розпочинається лише після завершення попередньої (рисунок 1.1).


Рисунок 1.1 – Життєвий цикл моделі Waterfall [28]

Такий підхід до розробки є доволі простим та прозорим, проте, паралельно занадто ідеалістичний та не практичний в умовах динамічного оточення. Модель Waterfall, зазвичай, використовують у проєктах із чітко визначеними вимогами, що не змінюються протягом реалізації проєкту. Здебільшого це проєкти машинобудівної інфраструктури, або державні проєкти, які потребують чіткого документування, розрахунку бюджету та аналізу всіх можливих ризиків. Якщо брати до уваги IT-індустрію, то розробка програмного продукту за моделлю Waterfall буде доречною, якщо мова йтиме про однотипні інформаційні системи (ІС), ІС із складними обчисленнями, ІС, що працюють у реальному часі, або ІС, до яких чітко та у повній мірі сформулювати усі вимоги на стадії планування.

До переваг такої моделі можна віднести:

- зрозуміла та чітка послідовність розробки програмного забезпечення, що, в свою чергу, знижує поріг входження команд на проєкт;
- стабільність, оскільки на етапі затвердження вимог та документації проєкт стає незмінний, протягом усього процесу розробки;
- можливість оцінити вартість та часові рамки ще до початку розробки проєкту;
- можливість легко відслідкувати процес впровадження проєкту, матеріальні ресурси, завдяки чіткій документації та суворій поетапності процесу розробки.

Як результат, отримуємо вчасно розроблений в межах бюджету цілісний програмний продукт (ПП) з документацією на нього. Такий ПП володіє високою зручністю впровадження та супроводу.

Проте, як вже зазначалось, цей підхід до розробки програмного забезпечення в умовах сучасного динамічного середовища у більшості випадків є не достатньо ефективним. Найпоширенішими проблемами процесу розробки на основі традиційних методів є:

- неможливість зміни вимог безпосередньо під час розробки;
- відсутність чіткого розподілення обов'язків за виконану роботу і її результат;
- наявність безперервного потоку «дрібних» додаткових завдань, які відволікають розробників і менеджерів від основної роботи;
- як висновок, невідповідність часовим рамкам, збільшення бюджету, втрата якості.

Проблема традиційного підходу до реалізації проєктів розробки програмного забезпечення у тому, що виконання триває довше, ніж очікувалось, витрати виявляються більшими, ніж закладалось у бюджеті і часто не досягають очікуваних результатів. Традиційні підходи до проєкт менеджменту беруть за основу співвідношення часу, витрат та обсягу робіт, що формують так званий «трикутник проєкт менеджменту» (рисунок 1.2), який у своїй книзі «Brilliant agile project management» Роб Коул та Едвард Скотчер називають «Бермудський трикутник» [65].

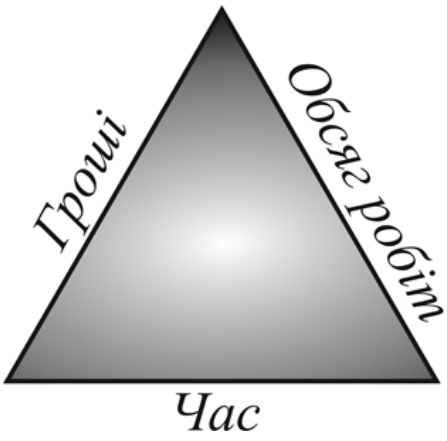



Рисунок 1.2 – Трикутник проєкт менеджменту [28]

Через жорсткі зв’язки, які лежать в основі цього трикутника, не можливо усунути одну з його сторін, не вплинувши на інші. Зміна будь-яких параметрів завжди матиме наслідки під час реалізації проєкту. Здебільшого це відбувається тоді, коли вносяться зміни до проєкту, скорочується час або бюджет. Проєкт-менеджерам дуже важко зберігати ці три складові в рівновазі і при цьому забезпечувати всі бажання клієнта. У результаті ми отримуємо або неякісний продукт, випущений вчасно у межах свого бюджету, або дійсно вартісний результат, ціна якого значно перевищуватиме початкову, а на часові проміжки вже не звертатимуть увагу.

Під час розробки продукту, використовуючи гнучкі методології важливим є лише одне – якість продукту. Методологія Agile відходить від традиційної одержимості термінами та бюджетами, зосереджуючись у першу чергу на тому, чого хоче клієнт, або – чого він дійсно потребує [27].

Рух Agile бере початок з концепції Lean – техніки «ощадливого виробництва», котра широко застосовується у автомобільній індустрії. Спершу методологію гнучкого управління використовували в галузі інформаційних технологій (IT). Тому не дивно, що саме у лютому 2001 року сімнадцятьма фахівцями, які зібрались на гірськолижному курорті The Lodge at Snowbird у штаті Юта, щоб обговорити принципи розробки програмного забезпечення, був опублікований «Маніфест гнучкої розробки програмного забезпечення», який раз і назавжди змінив підходи до процесу розробки та створення нових проєктів не тільки у галузі IT, а і у всіх інших сферах життєдіяльності.

«Маніфест гнучкої розробки програмного забезпечення» твердить, що [1]:

- люди та взаємодія важливіші за процеси та інструменти. Agile робить акцент на людській комунікації, командній роботі та розвитку розробників. Він говорить, що в першу чергу у команді повинен бути дух співпраці та взаємодопомоги. А дотримання жорстких правил та процесів швидше пасуватиме одинакам та диктаторам, які звикли працювати у одному стилі та не бажають змінюватись;

- працююче програмне забезпечення важливіше за вичерпну документацію. На думку Agile-спеціалістів, не варто витрачати дорогоцінний час та гроші на написання документації. Набагато краще втілити його у розробку готового функціоналу продукту. Це допоможе замовнику набагато швидше зрозуміти цінність та перспективи розвитку проєкту;
- співпраця із клієнтом набагато важливіша за обговорення умов контракту. Гнучка методологія Agile приділяє велику увагу спілкуванню із замовником. Нерідко буває так, що клієнт сам до кінця не знає, які саме функції повинен реалізовувати продукт та який термін та бюджет потрібні для його виконання. Дуже важко наперед передбачити всі умови створення проєкту. Тому саме тісна співпраця замовника із командою допомагає випустити дійсно вартісний продукт, який задовольнятиме всі його потреби;
- готовність до змін важливіше за дотримання плану. Agile методологія вважає зміни невід'ємною частиною розвитку хорошого проєкту. Втілені під час розробки ідеї можна реалізувати значно швидше, а, отже, і тестувати їх можна набагато раніше.

Насправді, під час створення проєктів важливо зрозуміти одне – замовник не хоче кращого управління. Він хоче кращого кінцевого продукту. Методологія гнучкого управління Agile спрямована на це [27]. Не важливо, які саме техніки та процеси ви використовуєте для досягнення кращого результату, не потрібно зосереджуватись на самих методах. Результат важливіший за шляхи, якими його досягнуто.

Розглянемо, чим відрізняється процес створення продукту на основі гнучких методологій від традиційних. Почнемо з того, що більшість гнучких методологій направлені на те, щоб мінімізувати ризики шляхом ітеративної розробки проєкту (рисунок 1.3).

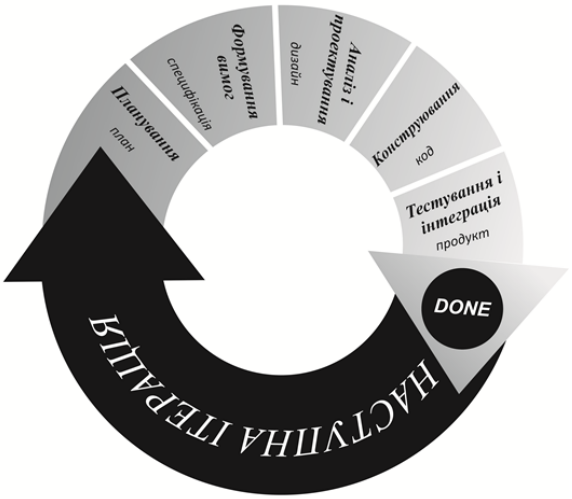



Рисунок 1.3 – Життєвий цикл Agile методології

Кожна ітерація це міні-проєкт, який включає в себе всі процеси розробки починаючи від планування і завершуючи випуском готової функціональності. Зазвичай, вони тривають два-три тижні (все залежить від того, який темп обере команда розробників) і закінчуються ретроспективою, тобто обговоренням того, що було зроблено та що можна зробити краще.

На стадії планування проєкту Agile починає із визначення необхідного мінімуму й працює вже з ним. Цей мінімум так і називається – мінімально життєздатний продукт (minimum viable product, MVP) або мінімальний набір функціональності (minimum feature set, MFS). На практиці мінімально життєздатний продукт вже відповідає основним бізнес вимогам проєкту і може бути проданий. Такий підхід зменшує часові та матеріальні витрати, необхідні на створення проєкту. Він може стати основою для більш складного і функціонального бізнес-рішення до того ж його набагато краще і легше тестувати. А підсумки тестування виявляються більш продуктивними, адже ґрунтуються на реальних фактах [26].

Не менш важливим є залучення розробника на всіх стадіях розробки проєкту. В Agile прийнято у кожну команду включати одну людину, зі сторони бізнесу, таку людину зазвичай називають власником продукту. Власник продукту представляє інтереси бізнесу і кінцевого користувача, він точно знає що саме потрібно людям, а не як це реалізувати. Основна його функція – донести до

команди завдання, які мають бути реалізовані з точки зору бізнесу, а команда, у свою чергу, реалізує це із технічної сторони. Для максимального ефекту замовник і команда розробників завжди повинні працювати в парі. Тільки таким чином можна створити дійсно якісний продукт, який задовольнятиме всі бажання клієнта.

Після того, як визначене практичне бачення проєкту, команді потрібно якомога детальніше записати вимоги до продукту. Такий перелік вимог у Agile називають беклогом. На відміну від звичного для традиційних моделей технічного завдання, беклог формує список важливих ідей для бізнесу. В цьому підтверджується орієнтація гнучких методологій на результат, а не на процеси. Команда бачить беклог і вже після того вирішує які дії, тобто технічні рішення їй потрібно виконати для задоволення конкретної вимоги, в свою чергу, виконання лише технічного завдання (формування якого властиве для традиційних моделей) не завжди означає одержання бажаного продукту.

Отож, до переваг використання Agile методології можна віднести:

- мінімізацію ризиків, завдяки гнучкій системі внесення змін;
- роботу на результат (вкінці кожної ітерації замовник отримує готовий продукт);
- зменшення документації (завдяки живому спілкуванню);
- можливість змінювати проєкт у ході розробки.

Пропри все, гнучкі методології розробки програмного забезпечення мають свої недоліки. Зокрема це:

- складність підрахунку кінцевої суми та часового проміжку проєкту;
- підвищені вимоги до кваліфікації і досвіду команди;
- постійні зміни, які можуть призвести до того, що проєкт ніколи не дійде до фінальної версії.

Порівняння традиційної методології (Waterfall ) та гнучкої методології (Agile) наведено у таблиці 1.1.

Таблиця 1.1

Порівняння традиційної методології (Waterfall ) та гнучкої методології (Agile)

| **Опис** | **Waterfall** | **Agile** |
|---|---|---|
| Робота над проєктом чітко спланована та послідовно виконується | так | ні |
| Внесення змін до проєкту в процесі розробки | ні | так |
| Сталий темп процесу розробки | ні | так |
| Тестування ПП в процесі розробки | ні | так |
| Управління директивне | так | ні |
| Контроль за роботою команди, що працює над розробкою | так | ні |
| У команді можлива взаємозаміна, розподілене лідерство | ні | так |
| Команда співпрацює в процесі розробки із замовником | ні | так |

Узагальнюючи вищенаведене можна стверджувати, що при організації життєвого циклу проєкту згідно з гнучкими методологіями розробниками значно швидше буде здійснено представлення замовнику прототипу ПП, його тестування та адаптація даного ПП до мінливих потреб ринку та вимог замовника і, як результат, значно швидша доставка готового ПП на ринок.

## 1.2. Специфіка та класифікація безпеко-орієнтованого програмного забезпечення (на прикладі сервісів для ДСНС)

Спеціалізоване програмне забезпечення (також відоме як нішеве, галузеве або вертикальне) — це тип програмного забезпечення, яке розроблене та призначене для вирішення конкретних, вузькоспрямованих завдань або для автоматизації процесів у певній галузі, ніші ринку, організації чи відділі. На відміну від універсального (горизонтального) ПЗ, яке підходить для широкого кола користувачів (наприклад, текстові програми або електронні таблиці), спеціалізоване ПЗ орієнтоване на потреби однієї галузі (наприклад, медицина, будівництво, банківська справа) або функції (наприклад, управління складськими запасами, розрахунок траєкторій польоту). Одним із підвидів спеціалізованого програмного забезпечення є безпеко-орієнтоване програмне забезпечення.

Безпеко-орієнтоване спеціалізоване програмне забезпечення (БОСПЗ) — це підвид спеціалізованого програмного забезпечення (СПЗ), яке розроблене для задоволення вузькоспеціалізованих потреб оперативних формувань, таких як Державна служба України з надзвичайних ситуацій (ДСНС), або інших військових/оперативних структур.

До прикладу, як у воєнний час, що запроваджено на території України, так і у період функціонування оперативних служб у повсякденному режимі, окремі ІТ-підрозділи займаються розробкою програмного забезпечення, вбудованих систем та інших безпеко-орієнтованих сервісів (далі – БОС), що орієнтовані на підвищення ефективності функціонування оперативно-рятувальних служб.

Членами команд розробки таких БОС в оперативних формуваннях є кадрові працівники відповідних служб, які в міру своєї службової підготовки повинні поєднувати діяльність, пов’язану із розробкою означених сервісів, з іншими різновидами оперативної та/або службової діяльності. Тому, зважаючи на специфіку роботи і особливих учасників проєктних команд із розробки безпеко-орієнтованих сервісів, динаміка проєктного середовища набуває дещо іншого значення.

Динамічність тепер зосереджується не лише на обсягу робіт, а й у часі їх реалізації (рисунок 1.4). З першого погляду можна припустити можливість розробки БОС в подібних умовах за каскадною моделлю. Проте, таке припущення буде хибним, адже більшість БОС не мають чіткого терміну необхідних робіт, а функціонал зазнає динамічних змін, поправок у ході розробки.

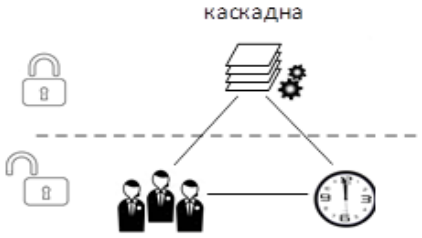



Рисунок 1.4 – Каскадна модель розробки програмного забезпечення [5]

За означених умов розробка безпеко-орієнтованих сервісів проєктними командами оперативних (військових) формувань буде набувати моделі, зображеної на рисунку 1.5.

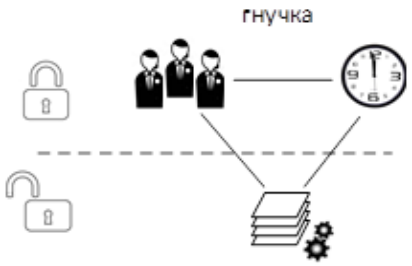



Рисунок 1.5 – Гнучка модель розробки програмного забезпечення [5]

Ця модель показує, що динамічні процеси з регламентованим переліком робіт, а також відсутність контролю (обмеження) часу на їх виконання може значно ускладнювати процеси своєчасного виконання роботи. Модель, за якою відбувається розробка БОС у порівнянні із класичними підходами до розробки програмного забезпечення зображена на рисунку 1.6.

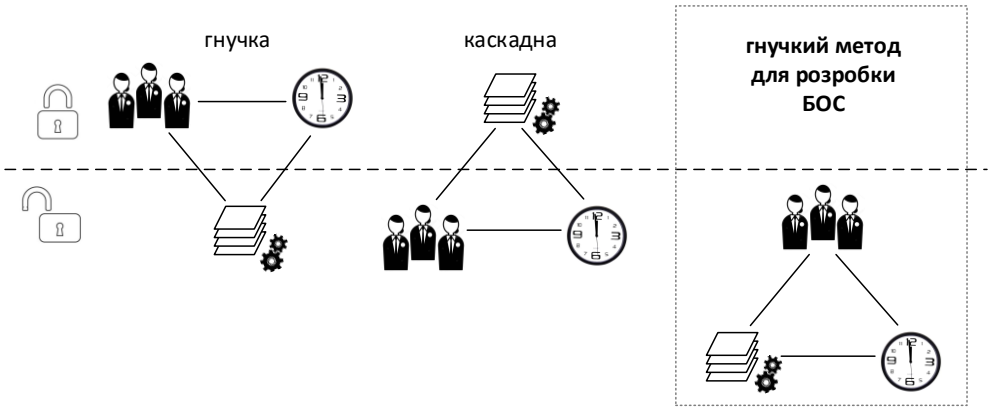



Рисунок 1.6 – Модель розробки БОС у порівнянні із класичними підходами до розробки програмного забезпечення [5]

Специфіка програмного забезпечення безпеко-орієнтованого спрямування породжує низку унікальних викликів, які ускладнюють застосування традиційних методологій управління життєвим циклом:

1. Динамічне та нестабільне середовище. Розробка часто відбувається в умовах, які є більш динамічними та менш стабільними, ніж у типових ІТ-командах. Це потребує можливості оперативного перепланування та гнучкого реагування на зміну пріоритетів. Кінцеві користувачі (фахівці ДСНС) працюють у сфері, де

потрібна негайна реакція, що впливає на вимоги до надійності та швидкості впровадження програмних рішень.

2. Специфіка проєктної команди. На відміну від класичних ІТ-компаній, членами команди розробки часто є кадрові працівники відповідних служб. Вони вимушені поєднувати діяльність, пов'язану з розробкою сервісів, зі своєю основною службовою та/або оперативною діяльністю. Цей фактор прямо впливає на доступність ресурсів та дотримання фіксованих термінів, характерних для класичних підходів.

3. Критичні вимоги до якості. БОПЗ належить до критичних систем, де надійність, точність та безпека даних є першочерговими вимогами. Будь-які помилки можуть мати серйозні наслідки для життєдіяльності та безпеки населення.

Узагальнюючи специфічні особливості програмного забезпечення безпеко-орієнтованого спрямування, порівняння класичних підходів до розробки ПЗ із БОС представлено у таблиці 2.

Таблиця 2. Порівняння класичних підходів до розробки ПЗ із БОС

| ***Опис*** | ***Waterfall*** | ***Agile*** | ***Гнучкий метод для розробки БОС*** |
|---|---|---|---|
| Робота над проєктом чітко спланована та послідовно виконується | так | ні | ні |
| Внесення змін до проєкту в процесі розробки | ні | так | так |
| Сталий темп процесу розробки | ні | так | ні |
| Тестування ПП в процесі розробки | ні | так | так |
| Управління директивне | так | ні | так |
| Контроль за роботою команди, що працює над розробкою | так | ні | так |
| У команді можлива взаємозаміна, розподілене лідерство | ні | так | так |
| Команда співпрацює в процесі розробки із замовником | ні | так | так |

З представлених результатів можна зробити висновок, що динамічність процесів планування обсягу робіт на визначений спринт, а також відсутність контролю (обмеження) часу на їх виконання, може ставати причиною не своєчасного або не якісного виконання проєкту. Саме тому виникає необхідність у

дослідженні існуючих методів оптимального планування часового ресурсу на розробку БОС, що відповідатимуть парадигмі гнучкого управління та дозволятимуть оперативно реагувати на відхилення від визначеного плану робіту.

### 1.3. Теоретичні передумови та концептуальний апарат запропонованої системи підтримки прийняття рішень

На основі проведеного в підрозділах 1.1 та 1.2 аналізу традиційних (Waterfall) та гнучких (Agile) методологій, а також визначення унікальної специфіки розробки безпеко-орієнтованого спеціалізованого програмного забезпечення (БОС) для оперативних формувань (зокрема, ДСНС), було встановлено, що існуючі підходи не забезпечують достатньої ефективності через динамічність середовища, особливості проєктної команди та відсутність контролю часу на виконання робіт. Враховуючи необхідність оперативного реагування на відхилення від плану та важливість адаптації під парадигму гнучкого управління, цей підрозділ присвячений формуванню теоретичних передумов та концептуального апарату запропонованої системи підтримки прийняття рішень. У ньому буде проаналізовано сучасні наукові праці щодо управління життєвим циклом критично важливих систем, обґрунтовано потребу у новому гнучкому підході, адаптованому до специфіки ДСНС, та представлено огляд математичного апарату, зокрема теорії графів, як основи для розробки методів оптимального планування часового ресурсу.

Аналіз останніх досліджень та публікацій свідчить, що питання вибору методології управління ІТ-проєктом є актуальним як для українських, так і зарубіжних вчених. Із появою «Маніфесту розробки програмного забезпечення» [1] все більше праць присвячено Agile методології та порівнянню її із традиційними методологіями управління життєвим циклом програмного забезпечення. Зокрема, теоретичні основи Agile висвітлені у роботах Роба Коула, Едварда Скотчера, Роберта Мартіна. В Україні питанням управління життєвим циклом програмного забезпечення займались такі вчені як С. Д. Бушуєв, І. Ю. Лєбєдєва, О. Б. Зачко, І. М. Якубенко, І. І. Оберемок, Т. О. Говорущенко, О. О. Павлова та інші.

Проте, вказані праці зосереджені, в основному, на розробці програмного забезпечення у класичних проєктних командах. Якщо ж говорити про розробку критично важливих сервісів (спеціалізованого програмного забезпечення безпекового спрямування), то підхід до управління життєвим циклом такого ПП відрізнятиметься, зважаючи на умови, проєктну команду та часові терміни впровадження такого продукту у життя.

Відомі наукові праці, у яких зроблені висновки про несумісність використання гнучких методологій та розроблення спеціалізованого програмного забезпечення [64; 74]. Тим не менш новіші наукові результати поставили під сумнів ці висновки, визначивши основні переваги, що виникають в процесі впровадження гнучких методів у розробку критично важливих систем, а саме: документація, оскільки вона не є необхідною для гнучкої розробки програмного забезпечення [100]; змінні вимоги, оскільки традиційні методи цьому не сприяють [100]; життєвий цикл розробки програмного продукту, оскільки дані проєкти не розробляються ні ітераційно, ні поступово (проблема із плануванням випуску програмного продукту) [99]; тестування, яке, у традиційній методології, проводиться тільки на завершальних стадіях розробки продукту [97].

Станом на сьогодні все частіше успішно використовують гнучкі методології управління життєвим циклом програмних систем у військовій [58, 59, 66, 99], залізничній [55], аерокосмічній [67, 110], медичній [101, 97] та інших сферах діяльності. Зокрема, у роботі [58] проведено дослідження на основі розробки системи командування та управління для Генерального штабу армії Італії із використанням гнучких методологій управління. Після тринадцяти п’ятитижневих спринтів розробники змогли надати готовий продукт, який відповідав усім функціональним та нормативним вимогам армії. Попри витрати, спрямовані на навчання та пристосування до нового методу розробки програмного забезпечення, витрати на саму розробку стали нижчими, ніж були раніше при використанні традиційних методів управління життєвим циклом програмних систем. У роботі [67] представлено результати реорганізації команд розробників у Scrum-команди в Agile Transformation Бразильського аеронавтичного обчислювального центру.

Зокрема був побудований симулятор польоту та розроблена система, яка контролює політ іноземних літаків у повітряному просторі Бразилії. Автори у роботі [97] описали традиційний підхід та виклики, з якими зустрічаються команди розробників в процесі створення програмного забезпечення для медичних установ, а також успішно впроваджені гнучкі практики до розробки в даній галузі. А у джерелі [73] наводяться рекомендації щодо відповідності програмного забезпечення міжнародним стандартам та документам регуляторних органів та при використанні гнучких практик agile у розробці програмного забезпечення для медичних установ.

Незважаючи на підтверджену можливість успішного впровадження гнучких практик у критично важливих доменах, зазначені дослідження не враховують повною мірою унікальний виклик, що постає перед оперативними формуваннями, а саме: вимушене поєднання членами команди основної службової/оперативної діяльності з процесом розробки БОС. Цей фактор, у поєднанні з динамічністю пріоритетів та відсутністю фіксованих часових обмежень, вимагає створення нового адаптивного механізму управління, що ґрунтуватиметься на гнучкій парадигмі, але забезпечуватиме ефективне планування часового ресурсу.

Автори у роботі [63] запропонували метод, заснований на розширенні підходу DevOps до безпекових систем. Даний автоматизований підхід можна легко зіставити з відомим методом гнучкої методології Scrum [62, 107]. У роботі [55] авторами також розширено цей метод, який надалі отримав назву Scrum for Safety (S4S). Він спрямований допомогти науково-дослідницьким групам розробляти безпекові рішення для залізничної галузі. У статті [110] йдеться про використання гнучкого підходу для забезпечення безпеки критично важливого вбудованого програмного забезпечення для космічного корабля NASA Orion. Йдеться, зокрема, про виклики, пов'язані з забезпеченням критичних функцій програмної системи та необхідність використання адаптивного підходу до забезпечення якості та валідації цих функцій у контексті еволюції методів розробки.

Автори у роботі [54] запропонували новий метод, що підтримує гнучку методологію розробки програмного забезпечення. Він базується на концепції MVC

і складається із трьох основних частин: модель, вигляд та контролер. Ідея полягає у тому, що замовник самостійно проектує користувацький інтерфейс. Це зменшує ризик відхилення або обмеженого використання програмного забезпечення і сприяє більш простій та безпомилковій розробці програмного забезпечення.

З огляду на проведений аналіз існує потреба у створенні нового гнучкого підходу до розробки спеціалізованого програмного забезпечення, котрий буде адаптований під специфіку роботи Державної служби України із надзвичайних ситуацій та корелюватиме із принципами гнучкої методології управління IT-проєктом. Для цього варто детально проаналізувати процес розробки програмного забезпечення на кожному етапі [79].

Станом на сьогодні існує велика кількість публікацій, присвячених процесу розробки програмних систем. Зокрема, у роботі [3] запропоновано залучати когнітивні технології на рівні експертних систем оцінки часу розробки, прийняття рішень щодо вибору архітектури, побудови програмних застосунків тощо. В науковій праці [109] автори розглянули процес розробки програмного забезпечення, який складається із чотирьох фаз, де архітектура програмного забезпечення є найголовнішою, оскільки забезпечує абстрактне представлення загальної структури програмної системи. У роботі [31] розроблено модель системи управління якістю процесу розробки програмного забезпечення на основі процесного підходу PDCA. У дослідженні [19] автори розробили методи якісного аналізу та кількісної оцінки ризиків розробки програмного забезпечення. У статті [105] використано теорію обмежень у контексті процесу розробки програмного забезпечення. В дослідженнях [28, 55, 56, 84, 88, 80, 114], основна увага приділена вибору методології для управління процесами розробки програмного забезпечення. В наукових роботах [21, 40, 94] описано процес створення сервісів, орієнтованих на забезпечення якості підготовки команд рятувальників у динамічному оточенні. Проте в згаданих роботах не зазначено, які саме моделі управління життєвим циклом розроблення програмних систем були використані при їх реалізації. Аналіз та математичне моделювання окремих етапів життєвого циклу спеціалізованого програмного забезпечення дозволили автоматизувати розробку безпеко-

орієнтованих сервісів та удосконалити процес управління життєвим циклом такого програмного забезпечення.

Зважаючи на основну мету роботи, варто також загадати про наукові праці, в яких автори більш детально досліджували етап планування в управлінні життєвим циклом розробки програмного забезпечення. Зокрема, у роботі [57] доведено, що планування займає центральне місце у Agile проектах, а всі інші сфери обертаються навколо нього. У роботі [60] також доведено важливість процесу планування та експертної оцінки завдань під час покер планування, а у науковій праці [61] вказано на важливість використання «дошки завдань» в процесі планування. В науковому джерелі [68] розглянуті обов'язкові кроки, які слід зробити, щоб отримати максимальну віддачу від гнучкої розробки програмного забезпечення. В [69] запропоновано використовувати інструмент графічного моделювання для прийняття рішень у міждисциплінарному дослідницькому співробітництві.

Оскільки у даній роботі запропоновано використовувати мережеві графи на етапі планування життєвого циклу спеціалізованого програмного забезпечення, варто також розглянути їх застосування у сучасній науці.

Теорія графів – спеціальний розділ математики, який вирішує велику кількість наукових та прикладних завдань. Зокрема, розв'язання задач із кібернетики, програмування, штучного інтелекту, логістики. Із використанням графів можна швидко та якісно дослідити різні електричні, транспортні або комп'ютерні мережі, системи електронної комунікації, водопостачання або електрифікації. Чіткий та зрозумілий механізм побудови графів дозволяє з легкістю змоделювати та інтерпретувати потрібні процеси (події чи явища), реалізувати їх наочне представлення без необхідності фізичного проектування.

Станом на сьогодні існує велика кількість різноманітних алгоритмів та методів обходу графів. Більшість із них спрямовані на визначення найкоротшого шляху. Це, зокрема, алгоритми Дейкстри, Флойда-Уоршела, Беллмана-Форда. У роботах [10, 33, 48, 104] описано основні сфери застосування даних алгоритмів. Здебільшого це логістика, комп'ютерні мережі, системи електронної комунікації та інші. Завдання цих алгоритмів полягає у знаходженні оптимального шляху між

заданими вершинами. Існує також ряд алгоритмів, спрямованих на знаходження пропускної здатності мережі. Це, зокрема, алгоритми Форда-Фалкерсона, Едмондса-Карпа, Дініца. У роботах [32, 37, 69] описане практичне застосування даних алгоритмів. Основною їх задачею є знаходження такого потоку, щоб його величина була максимальна. Здебільшого вони використовуються для оцінки завантаження потокової чи транспортної мережі.

Математичний апарат теорії графів широко використовується у галузі штучного інтелекту. Існує окремий клас методів глибинного навчання (графові нейронні мережі), призначений для формулювання логічних висновків на основі даних, описаних цими графами [118]. Бібліотека машинного навчання TensorFlow використовує графи потоків даних для здійснення чисельних обчислень. Вузли графів представляють собою математичні операції, а грані – багатовимірні масиви даних (“тензори”), що “протікають” між вузлами [7]. У статті [13] запропоновано використовувати векторний скелет (граф) для побудови алгоритму розпізнання символів для систем штучного інтелекту.

Теорія графів широко застосовується у криптографії та стеганографії. У роботі [52] представлено новий стеганографічний підхід до приховування даних у растровому зображенні. Запропонований підхід розподіляє зображення на словник пар «ключ-значення» у стилі JSON і використовує метадані графа для визначення розташування різних частин і позицій зображення корисної інформації у всьому кластері покритого зображення. Теорія графів активно використовується і у маркетинговій справі, зокрема, для побудови ефективної реклами. Якщо перші пошукові сервіси використовували ключові слова для пошуку інформації, таким чином створювали ієрархію сторінок за кількістю переглядів, то пошукова система Google повністю змінила свою концепцію, цим самим стала однією із монополістів на ринку пошукових систем. Дана система для надання рангу сторінці використовує сімейство алгоритмів RankPage. Алгоритм заснований на методі графів виявлення ступеня важливості сторінки через кількість цитувань [6] (до речі, підґрунтям для цього став алгоритм для визначення наукометричних рейтингів журналів).

Також відома низка робіт [75, 81, 82], присвячених використанню мережевих графів у процесі планування проекту. Зокрема, авторами у роботі [75] обрано метод розв'язання задачі планування, заснований на застосовуванні методу мережевого планування. Він базується на ідеї оптимізації критичного шляху із залученням додаткових обмежених коштів. Цей підхід не корелює із розробкою спеціалізованого програмного забезпечення. У статті [81] описано процес розробки комп'ютерної програми розв'язання задач мережевої оптимізації. Проте розрахунок найкоротших шляхів проходить за алгоритмом Дейкстри, який не є універсальним і не адаптований для мережевого планування процесу розробки програмного забезпечення. В основу дослідження [82] закладено методи мережевого планування PERT, засоби елементів теорії графів та методу діаграм Ганта. Підхід актуальний для каскадної моделі управління, якому притаманне послідовне виконання завдань та визначений час виконання.

### 1.4. Висновки

За результатами розділу можна зробити такі висновки:

1. Встановлено, що традиційний підхід до розробки програмного забезпечення Waterfall є неефективним в умовах динамічного середовища оперативних формувань, тоді як Agile, з його гнучкістю, ітеративністю та фокусом на якості продукту, є більш перспективним. Проте, чиста Agile-модель має обмеження щодо складності підрахунку кінцевої вартості та часового проміжку, що критично для спеціалізованих проєктів.

2. Визначено унікальну специфіку безпеко-орієнтованого спеціалізованого програмного забезпечення (БОС) для оперативних формувань (ДСНС). Головні відмінності полягають у динамічному та нестабільному середовищі, критичних вимогах до надійності та якості, а також у складі проєктної команди, де кадрові працівники поєднують розробку зі своєю основною службовою діяльністю. Це призводить до порушення сталого темпу роботи та відсутності контролю часу на виконання, що робить класичні методології недостатньо ефективними.

3. З огляду на виявлену недостатню ефективність існуючих підходів для розробки БОС, обґрунтовано необхідність створення нового адаптивного механізму управління. Цей механізм має базуватися на принципах гнучкої методології (Agile), а також бути адаптованим до специфіки ДСНС та націлений на ефективне планування часового ресурсу. У якості концептуального апарату для розробки методів оптимального планування запропоновано використовувати математичний апарат теорії графів, зокрема мережеві графи, для побудови імітаційних моделей – мережі Петрі, а для оцінки обсягів робіт – математичні множини.

# РОЗДІЛ ІІ. МОДЕЛЮВАННЯ ПРОЦЕСУ УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В УМОВАХ ДИНАМІКИ

## 2.1. Математична модель процесу розробки спеціалізованого програмного забезпечення

Розробка математичної моделі процесу створення спеціалізованого програмного забезпечення є ключовим етапом, що дозволяє не лише аналітично описати життєвий цикл продукту, але й здійснити оцінку обсягів робіт на кожному етапі, що є важливим для ефективного планування та управління розробкою програмного забезпечення.

Наявні дослідження у сфері управління життєвим циклом програмних систем, зосереджуючись на теоретичних етапах розробки, недостатньо висвітлюють можливості їх автоматизації. Це зумовлює необхідність створення математичної моделі, здатної описати процес розробки спеціалізованих систем, що стане основою для подальшої оптимізації ресурсної складової управління життєвим циклом прогрмного забезпечення. Така модель дозволить підвищити якість продукту, скоротити час та матеріальні витрати на розробку, а також загалом поліпшити управління життєвим циклом безпеко-орієнтованого програмного забезпечення. Для її побудови залучено понятійний апарат теорії множин та визначено ключові взаємозв’язки між окремими етапами розробки.

Побудову геометричної моделі життєвого циклу спеціалізованих програмних систем проведемо у певному порядку. На рисунку 2.1 поданий універсум <U> з елементами у вигляді множини, які імітують етапи розроблення програмного забезпечення.

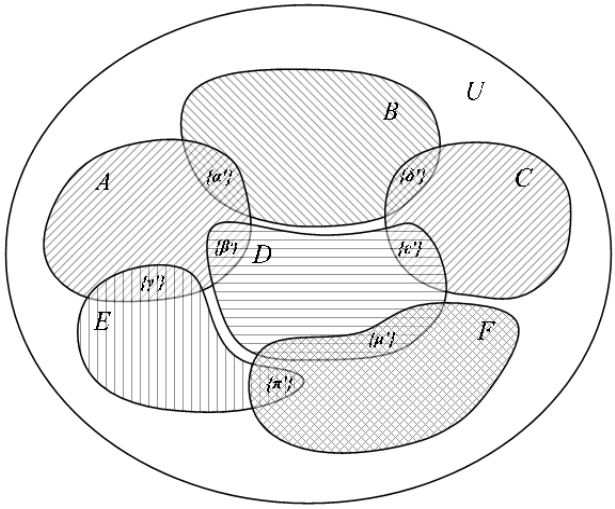



Рисунок 2.1 – Геометрична модель життєвого циклу спеціалізованих програмних систем у вигляді універсуму <U>

Вміст універсуму можна описати так:

$$U = \{A, B, C, D, E, F\}, \tag{2.1.}$$

де $\{A\}$ – множина робіт з аналізу предметної області та формулювання вимог;

$\{B\}$ – множина робіт з проєктування архітектури програми та UX-дизайну;

$\{C\}$ – множина робіт з реалізації програмної системи в кодах;

$\{D\}$ – множина робіт з тестування програмної системи;

$\{E\}$ – множина робіт з впровадження програмної системи;

$\{F\}$ – множина робіт з супроводу програмної системи в процесі експлуатації.

В універсумі $<U>$ існують елементи, які належать перетину множин:

$$A \cap B; A \cap D; A \cap E; B \cap C; C \cap D; D \cap F; E \cap F. \tag{2.2}$$

У результаті виконання робіт універсуму <U> отримуємо готовий програмний продукт. Позначимо $R(X)$ – результат виконання множини *X*, зокрема:

$$\{\alpha'\} = R(A \cap B), \tag{2.3}$$

де $\alpha'$ – визначені вимоги до програмного забезпечення на основі аналізу предметної області і формулювання системних вимог;

$$\{\beta'\} = R(A \cap D), \tag{2.4}$$

де $\beta'$ – вимоги до тестування програмного забезпечення, які визначаються на етапі аналізу предметної області та формулювання системних вимог (перевірка відповідності цим вимогам);

$$\{\gamma'\} = R(A \cap E), \tag{2.5}$$

де $\gamma'$ – вимоги до впровадження програми, які визначаються на етапі аналізу предметної області і формулювання системних вимог (до прикладу вибір платформи для розгортання продукту, визначення етапів розгортання продукту на боці замовника, визначення етапів навчання користувачів тощо);

$$\{\delta'\} = R(B \cap C), \tag{2.6}$$

де $\delta'$ – програмний код, що реалізує функціонал програмної системи відповідно до специфікації та поданої архітектури;

$$\{\varepsilon'\} = R(C \cap D), \tag{2.7}$$

де $\varepsilon'$ – результати тестування програмного коду на відповідність специфікації та результати опрацювання журналу помилок (допомагає виявляти проблеми в роботі програмної системи та їх природу);

$$\{\mu'\} = R(D \cap F), \tag{2.8}$$

де $\mu'$ – технічні звіти за результатами тестування, які містять інформацію про виявлені проблеми, шляхи їх вирішення та рекомендації щодо тестування і виправлення недоліків в процесі експлуатації програмної системи;

$$\{\pi'\} = R(E \cap F), \tag{2.9}$$

де $\pi'$ – вимоги до процесів підтримки програмного забезпечення.

Для глибшого аналізу процесу розроблення програмного забезпечення розглянемо кожну із множин універсуму <*U*>. На рисунку 2.2 зображена модель множини *A*, яка відтворює етап аналізу предметної області і формулювання системних вимог (постановки завдання) до програмного продукту.

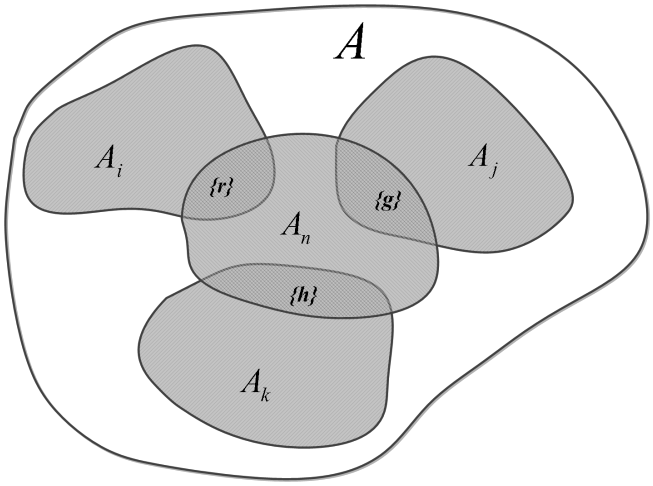



Рисунок 2.2 – Множина робіт з аналізу предметної області і формулювання вимог

Множина $A$ у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$A = \{A_i, A_j, A_k, A_n\}, \tag{2.10}$$

де $A_i$ – множина робіт із оцінки вимог та потреб стейкхолдерів;

$A_j$ – множина робіт з визначення обмежень та вимог до функціональності;

$A_k$ – множина робіт з визначення фінансових та ресурсних обмежень проєкту;

$A_n$ – множина робіт із створення документації проєкту.

В множині *A* між підмножинами існують елементи, які належать їх перетину:

$$A_i \cap A_n; A_j \cap A_n; A_k \cap A_n. \tag{2.11}$$

У результаті виконання робіт множини *A* отримуємо сформульований аналіз предметної області і системних вимог до програмного продукту. Виконання робіт, які належать перетину підмножин множини *A,* призводять до отримання нових результатів :

$$\{r\} = R(A_i \cap A_n), \tag{2.12}$$

де $r$ – сформоване технічне завдання проєкту;

$$\{g\} = R(A_j \cap A_n), \tag{2.13}$$

де $g$ – сформована специфікація програмного забезпечення;

$$\{h\} = R(A_k \cap A_n), \tag{2.14}$$

де $h$ – сформований фінансовий та ресурсний план проєкту.

Наступний етап життєвого циклу програмного забезпечення – проєктування архітектури програми та UX-дизайну інтерфейсу. На рисунку 2.3 зображена графічна модель цього етапу у вигляді множини *B*.

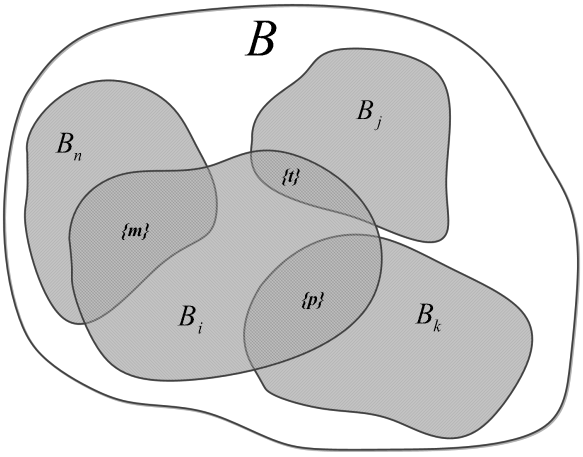



Рисунок 2.3 – Множина робіт з проєктування архітектури програми та UX-дизайну

Множина *B* у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$B = \{B_i, B_j, B_k, B_n\}, \tag{2.15}$$

де $B_i$ – множина робіт із створення архітектури системи;

$B_j$ – множина робіт із вибору програмних технологій;

$B_k$ – множина робіт із побудови та написання алгоритмів;

$B_n$ – множина робіт з розробки дизайну та моделі людино-машинної взаємодії.

У множині $B$ між підмножинами існують елементи, які належать до їх перетину:

$$B_i \cap B_j; B_n \cap B_i; B_k \cap B_i. \tag{2.16}$$

У результаті виконання робіт множини $B$ отримуємо спроєктовану архітектуру програми та UX-дизайн інтерфейсу. Виконання робіт, які належать перетину підмножин множини $B$, призводять до отримання нових результатів :

$$\{t\} = R(B_j \cap B_i), \tag{2.17}$$

де $t$ – архітектура структурних елементів програмної системи у вигляді діаграми класів та діаграми об’єктів;

$$\{m\} = R(B_n \cap B_i), \tag{2.18}$$

де $m$ – архітектура структурних елементів програмної системи у вигляді діаграми прецедентів, станів та компонентів (прототип програми);

$$\{p\} = R(B_k \cap B_i), \tag{2.19}$$

де $p$ – архітектура структурних елементів програмної системи у вигляді діаграм взаємодії та розміщення.

Наступний етап життєвого циклу програмного забезпечення – реалізація програмного продукту в кодах (імплементація програмної системи). На рисунку 2.4 зображена геометрична модель даного етапу у вигляді множини *C*.

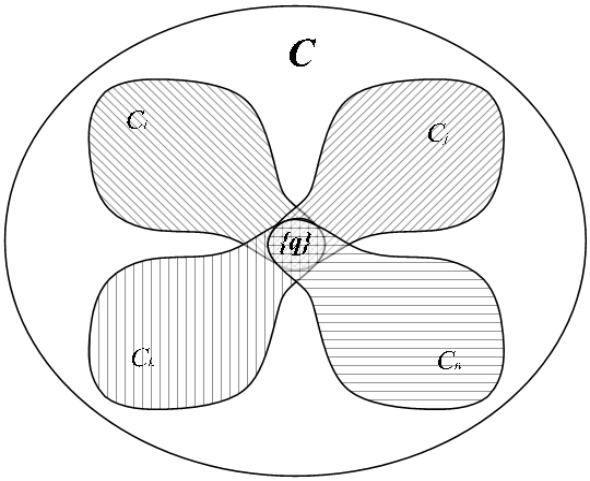



Рисунок 2.4 – Множина робіт з реалізації програмної системи в кодах

Множина *C* у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$C = \{C_i, C_j, C_k, C_n\}, \tag{2.20}$$

де $C_i$ – множина робіт з реалізації програмної системи в кодах;

$C_j$ – множина робіт з оптимізації та тестування коду (Unit tests);

$C_k$ – множина робіт з виявлення та усунення дефектів, що виникають в процесі тестування (Unit Tests);

$C_n$ – множина робіт із розробки документації до коду програми.

В множині С між підмножинами існують елементи, які належать їх перетину:

$$C_i \cap C_j; C_i \cap C_k; C_i \cap C_n; C_j \cap C_k; C_j \cap C_n; C_n \cap C_k. \tag{2.21}$$

У результаті виконання робіт множини *C* отримуємо реалізацію програмної системи в кодах. Виконання робіт, які належать перетину підмножин множини *C,* призводять до одержання нового результату :

$$\{q\} = R(C_i \cap C_j \cap C_k \cap C_n), \tag{2.22}$$

де $q$ – готовий програмний модуль із супровідною документацією.

Черговий етап життєвого циклу програмного забезпечення – тестування програмного продукту (QC/QA). На рисунку 2.5 зображена геометрична модель цього етапу у вигляді множини *D*.

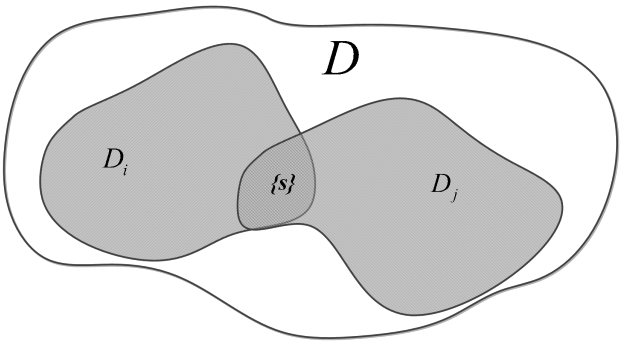



Рисунок 2.5 – Множина робіт із тестування програмної системи

Множина *D* у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$D = \{D_i, D_j\}, \tag{2.23}$$

де $D_i$ – множина робіт із виконання тестів на функціональність, продуктивність, безпеку тощо;

$D_j$ – множина робіт із перевірки відповідності вимогам технічного завдання.

В множині *D* між підмножинами існують елементи, які належать їх перетину:

$$D_i \cap D_j. \tag{2.24}$$

У результаті виконання робіт множини *D* отримуємо протестовану програмну систему. Виконання робіт, які належать перетину підмножин множини *D,* призводять до отримання нового результату:

$$\{s\} = R(D_i \cap D_j), \tag{2.25}$$

де *s* – пройдені тест-кейси (test cases) або/та звіти про помилки (bug reports).

По завершенні етапу тестування життєвий цикл програмного забезпечення переходить до етапу впровадження програмного продукту шляхом його розгортання на боці клієнта (видача замовнику). На рисунку 2.6 відтворена геометрична модель цього етапу у вигляді множини *E*.

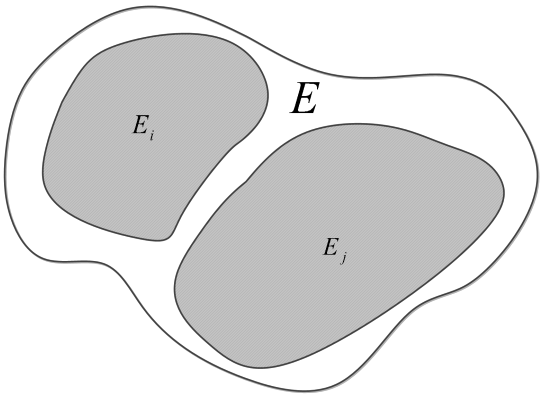



Рисунок 2.6 – Множина робіт із впровадження програмної системи

Множина *E* у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$E = \{E_i, E_j\}, \tag{2.26}$$

де $E_i$ – множина робіт з підготовки програмного середовища для розгортання програмного продукту;

$E_j$ – множина робіт із навчання користувачів.

У результаті виконання робіт множини *E* отримуємо впроваджену програмну систему на боці клієнта. Виконання робіт, які належать перетину підмножин множини *E*, призводять до отримання наступного результату:

$$R(E_i \cap E_j) = \varnothing. \tag{2.27}$$

Нарешті останній етап життєвого циклу програмного забезпечення – супровід програми в процесі експлуатації (підтримка програмного забезпечення) до моменту завершення експлуатації продукту. На рисунку 2.7 зображена геометрична модель цього етапу у вигляді множини *F*.

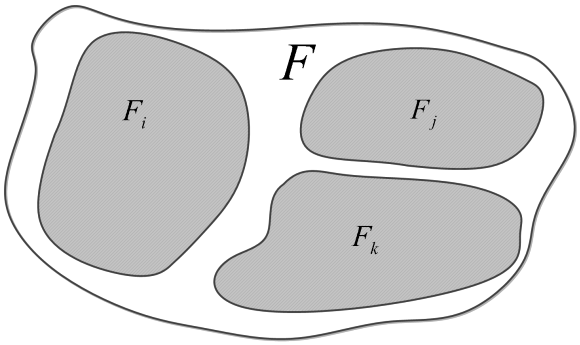



Рисунок 2.7 – Множина робіт із супроводу програмної системи під час експлуатації.

Множина *F* у відтвореній моделі містить підмножини та може бути представлена у вигляді:

$$F = \{F_i, F_j, F_k\}, \tag{2.28}$$

де $F_i$ – множина робіт із виправлення помилок та виявлення нових дефектів;

$F_j$ – множина робіт із надання користувачам необхідної допомоги та підтримка;

$F_k$ – множина робіт із доповнення програмного продукту новим функціоналом, що концептуально відтворює усі попередньо описані етапи ЖЦ розроблення програмних систем.

У результаті виконання робіт множини $F$ отримуємо супровід програмної системи під час експлуатації. Виконання робіт, які належать перетину підмножин множини $F$ призводять до отримання таких результатів:

$$\begin{aligned} R(F_i \cap F_j) &= \varnothing; \\ R(F_i \cap F_k) &= \varnothing; \\ R(F_j \cap F_k) &= \varnothing. \end{aligned} \tag{2.29}$$

Приведені геометричні та математичні моделі множин із використанням понятійного апарату теорії множин допомагають узагальнити структуру та описати усі етапи життєвого циклу програмного забезпечення на різних етапах (аналіз предметної області та формулювання системних вимог, проектування архітектури програми та UX-дизайну інтерфейсу, реалізація програми в кодах, тестування програми, впровадження програми, супровід програми під час експлуатації).

Математична модель дає підстави сформувати повну та чітку уяву про обсяги робіт, виконання яких покладене в основу успішної реалізації програмних систем, у тому числі безпеко-орієнтованого спрямування. Отримана модель дозволяє охарактеризувати окремі етапи розробки продукту та аналітично описати життєвий цикл розроблення програмного забезпечення, що є передумовою для пошуку шляхів автоматизації окремих процесів підтримки прийняття управлінських рішень.

## 2.2. Концептуальна модель процесу управління життєвим циклом спеціалізованого програмного забезпечення

Проведений у першому розділі літературний аналіз вказує на потребу створення нових методів управління життєвим циклом розроблення спеціалізованого програмного забезпечення (безпеко-орієнтованих сервісів), котрі адаптовані під специфіку роботи Державної служби України із надзвичайних ситуацій та корелюють із принципами гнучкої методології управління ІТ-проєктом. У даному підрозділі обґрунтовано як гнучкість може допомогти у інноваціях спеціалізованого безпеко-орієнтованого програмного забезпечення, щоб підвищити ефективність та надійність таких програмних продуктів, об’єднано основні концепції методів гнучкої методології управління життєвим циклом програмного забезпечення, зважаючи на специфіку розробки такого ПЗ для служби порятунку (зокрема основний акцент здійснено на гнучкий scrum метод) та розроблено концептуальну модель процесу управління життєвим циклом розроблення безпеко-орієнтованих сервісів, яка ґрунтується на гнучкому підході. Реалізація цієї моделі в умовах динаміки проєкту вимагає використання математичного апарату для забезпечення гнучкого планування.

Для розв’язання цієї потреби використаний математичний апарат теорії графів, адаптований до процесів мережевого планування шляхом оптимізації розрахунку його основних часових параметрів під динамічні умови проєкту. Запропонований нами підхід дозволить у реальному часі здійснювати перепланування завдань із спринта, враховуючи критично важливі задачі для створення мінімально-життєздатного продукту (mvp) програмної системи.

Тоді додаткове завдання для команди розробників після пріоретизаціїї та оцінки користувацьких історій у беклозі спринта полягає у визначенні попередніх користувацьких історій чи функцій, які мають бути виконані, оскільки існують такі задачі, виконання яких можливе тільки після завершення попередніх. У такому випадку процес розроблення безпеко-орієнтованих сервісів набуде такого вигляду (рисунок 2.9).

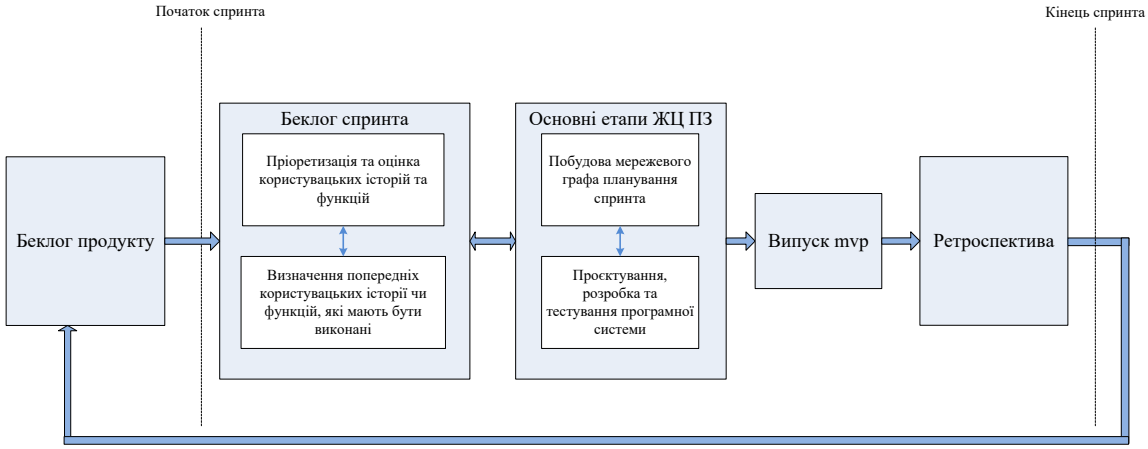



Рисунок 2.9 – Розроблена концептуальна модель процесу управління життєвим циклом БОС

Відповідно до розробленої нами концептуальної моделі управління БОС починається із формування беклогу продукту. У даному випадку беклог містить всі завдання (користувацькі історії) та функції, які необхідно виконати для розробки усієї програмної системи. Після формування беклогу весь процес розробки, як і в звичній Scrum команді, розбивається на ітерації (спринти), кожна з яких повинна завершуватись випуском готового функціоналу (mvp) та ретроспективою.

Зважаючи на специфіку розроблення безпеко-орієнтованих сервісів, беклог спринта формується наступним чином:

1. Пріоретизуються та оцінюються користувацькі історії та функції, які необхідно виконати за спринт.

2. Визначаються попередні користувацькі історії чи функції, які мають бути виконані, оскільки існують такі задачі, виконання яких можливе тільки після завершення попередніх.

Між цими пунктами існує двосторонній зв'язок, оскільки при зміні пріоритету користувацької історії може змінюватись також і попередня користувацька історія.

Після формування беклогу спринта відбувається розроблення самої системи. Для автоматизації етапу планування запропоновано побудувати мережевий граф планування спринта та обчислити його основні показники (критичний шлях, ранні та пізні терміни виконання подій, резерви на виконання завдань). Такий підхід

дозволить проєктній команді відслідковувати в режимі реального часу стан розробки системи та, за потреби, змінювати хід виконання завдань. Мережеве планування надасть можливість ефективно розподіляти час на розробку та автоматизувати процес визначення критично важливих функцій для випуску mvp.

Після планування реалізуються звичні етапи життєвого циклу розроблення програмного забезпечення такі як проектування, розробка та тестування програмної системи. На концептуальній схемі між плануванням та наступними етапами ЖЦ ПЗ показаний двосторонній зв’язок, оскільки залежно від процесу розробки, складності завдань, досвіду проєктної команди та інших чинників час на реалізацію може змінюватись, тому може виникнути потреба у перепланування мережевого графа. Двосторонній зв’язок існує також і між формуванням беклогу спринта та основними етапами ЖЦ ПЗ, оскільки залежно від визначеного критичного шляху під час мережевого планування, завдання та користувацькі історії беклогу спринта можуть змінюватись і навпаки.

Завершується спринт випуском готового продукту, який може бути використаний у службовій діяльності працівниками ДСНС та ретроспективою, на якій обговорюються шляхи його покращення та вносяться корективи на наступну ітерацію.

## 2.3. Імітаційні моделі обходу мережевого графа із застосуванням мереж Петрі

В працях [9, 10, 12] означено, що мережеві графи дають змогу відобразити роботи проєкту та зв’язки між ними. Такі графи зазвичай використовують в управлінні проєктами для довгострокового планування та оцінки стану виконання проєкту на поточному етапі. Найпопулярнішим алгоритмом мережевого планування є метод PERT (Project Evaluation and Review Technique). Ця мережа передбачає оцінку кожної роботи із використанням трьох тимчасових станів: оптимістичний, песимістичний та найбільш ймовірний час. Як відомо, мережевий граф дає змогу розрахувати ключові показники для знаходження критичного шляху виконання робіт, а також резерви часу для кожної роботи окремо. Цей метод добре

зарекомендував себе в управлінні масштабними проєктами, проте вивчення ефективності його застосування для короткострокового планування розробки програмного забезпечення у динамічних умовах досліджено не достатньо. Крім того, попередні наукові досягнення вказують на складність використання стандартних підходів проєктного менеджменту (гнучкі та каскадні) до управління процесом розроблення спеціалізованого програмного забезпечення у динамічному оточенні. Зважаючи на зазначене, в розділі поставлено мету моделювання процесів обходу мережевого графа для розрахунку ключових параметрів мережевої моделі, що дасть змогу швидко та якісно оцінити тривалість проєктних робіт в умовах динамічності ресурсів та змісту робіт. Моделювання дискретних процесів мережевого планування допоможе сформувати повноцінну уяву щодо застосування PERT-підходів для задач короткострокового планування проєктів з розроблення безпеко-орієнтованих сервісів, які володіють значним показником динамічності. Створення моделі є визначальним етапом у побудові алгоритмів обходу мережевого графа для оцінки тривалості виконання проєктних робіт. Своєю чергою розроблені алгоритми дадуть змогу надалі автоматизувати процеси короткострокового планування, а також швидко і якісно проводити переоцінку тривалості робіт за умови введення будь яких змін до змісту, обсягу, часу виконання окремих спринтів проєкту або складу команди розробників.

Для кращого розуміння об’єкта моделювання, процесу обходу мережевого графа для визначення його ключових параметрів, варто розглянути основні елементи мережевої моделі в контексті гнучкої методології управління IT-проєктами. Оскільки в мережевому плануванні основними елементами моделі є роботи та події, то в контексті Agile-технології: робота – це користувацька історія або функція; подія – початок або завершення будь якої роботи (користувацької історії). Події у мережевій моделі позначаються вершинами графа, а роботи – його ребрами. За одиницю вимірювання робіт в процесі моделювання буде використано універсальну відносну величину story points (далі – s.p.), яка враховує поєднання трудомісткості завдання, його складності та ризиків, які з ним пов'язані. Такий різновид оцінки дає змогу враховувати часові та ресурсні обмеження процесу

розроблення спеціалізованого програмного забезпечення залежно від досвіду команди розробників.

Для виконання процедури мережевого планування (побудови мережевого графа) необхідно [12]:

1. Пріоритезувати користувацькі історії та функції, які необхідно виконати.
2. Оцінити користувацькі історії та функції у s.p.
3. Визначити попередні користувацькі історії чи функції, які мають бути виконані, оскільки існують такі завдання, виконання яких можливе лише після завершення попередніх.

Наявність мережевого графа дає можливість сформувати уяву про орієнтовні обсяги робіт та їх черговість. Проте наявність лише самого графа не дає змоги оцінити тривалість робіт за проєктом. Побудова графа мережевої моделі в початковому стані не дасть змоги проводити оперативний перерозподіл обсягів робіт із подальшим визначенням тривалості їх виконання в динамічному оточенні. З цією метою математичний апарат мережевого планування володіє низкою методів розрахунку параметрів мережевого графа для часової оцінки процесів розробки специфічного програмного забезпечення на стадії планування, а саме: ранній термін виконання подій $T_r$; пізній термін виконання подій $T_l$; критичний шлях виконання робіт; часові резерви $R_{подій}$, $R_{робіт}$. В математичному відношенні визначення цих параметрів не передбачає жодної складності. Проте в алгоритмічному представленні, з метою подальшої автоматизації розрахунків, можуть виникати труднощі, пов’язані з коректністю роботи алгоритму, його безвідмовністю тощо. Це може бути зумовлене складністю самої мережевої моделі, наявністю великої кількості робіт, у тому числі фіктивних, наявністю великої кількості подій, наявністю або відсутністю взаємозв’язків між ними, розгалуженістю мережевої моделі тощо.

Зважаючи на це, процедуру обходу мережевого графа для розрахунку його ключових параметрів необхідно представити у вигляді уніфікованої імітаційної моделі. Універсальність моделі полягатиме у її спроможності відтворення

процедури обходу будь якого мережевого графа незалежно від його складності та конфігурації. І, найголовніше, уніфікована модель створить передумови для побудови відповідних алгоритмів та автоматизації процесів розрахунку параметрів мережевого графа.

Зважаючи на те, що мережевий граф та процедура його обходу є дискретною системою, для побудови імітаційної моделі процесів визначення параметрів $T_f$, $T_l$ обрано математичний апарат моделювання розподілених дискретних систем мережами Петрі. Мережі Петрі обрано з точки зору зручності імітаційного відтворення паралельних (розподілених) процесів, а також їх динамічної взаємодії в системі. Саме такими характеристиками володіють мережеві моделі, які досліджуються в попередніх роботах [9, 10, 12].

В дискретній моделі, описаній за допомогою мережі Петрі, події відображаються переходами t, умовами виконання яких є позиції P. На першому етапі реалізується визначення раннього терміну виконання подій. Цей параметр характеризує термін, раніше якого подія відбутись не може, та дає змогу контролювати початок виконання робіт, які не регламентовані порядковістю. Для першої події $T_{f(1)} = 0$, для всіх подальших подій мережевої моделі Tf визначається відповідно до методики.

Отже, відтворимо та опишемо модель процесу обходу мережевого графа з метою визначення раннього часу виконання робіт $T_f$ (рисунок 2.10).

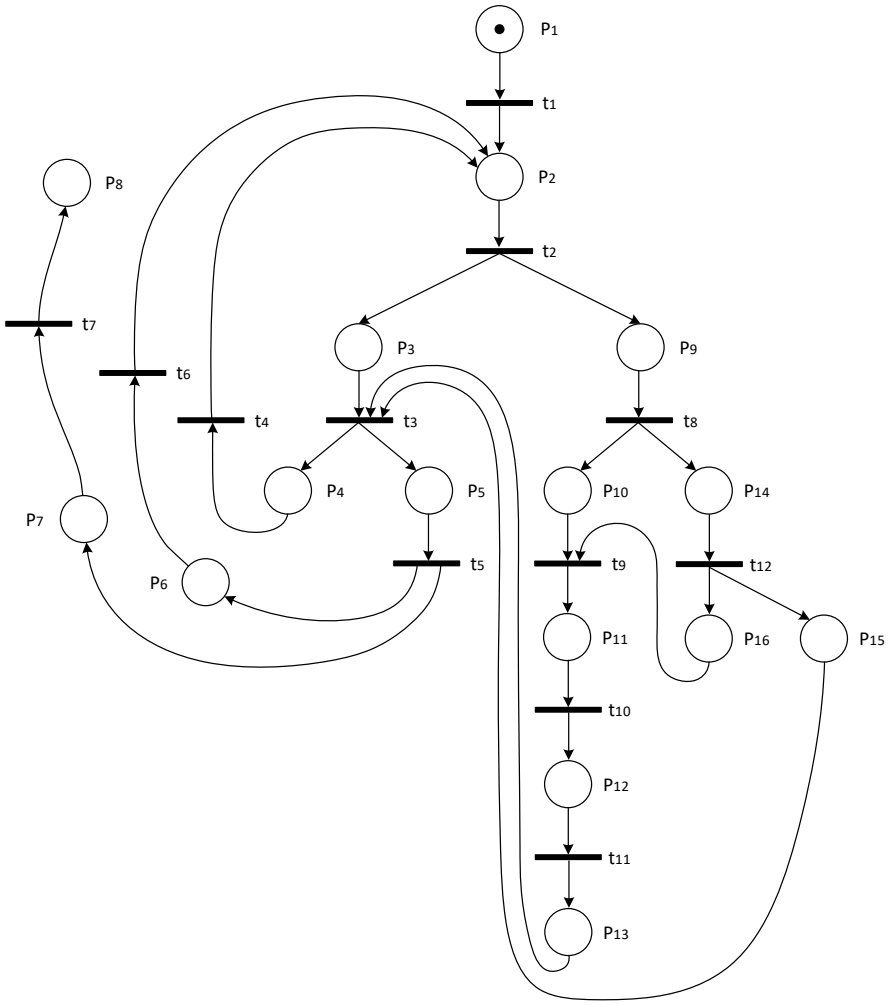



Рисунок 2.10 – Імітаційна модель процесу обходу мережевого графа з метою визначення раннього часу виконання робіт $T_f$

Процес обходу мережевого графа для розрахунку раннього терміну виконання подій починається з позиції (умови) $P_1$, яка відповідає за перевірку наявності початкових даних, а саме: індексу попередньої події $i = 0$; індексу наступної події $j = 1$; обсягу подій мережевої моделі $n$; множини робіт мережевої моделі $\{t_{i-j}\}$. Перехід (подія) $t_1$ відповідає за підготовку початкових параметрів циклічного обходу мережевого графа з такими значеннями: ранній термін виконання першої події $T_{f(1)} = 0$; $i$++; $j = i+1$. Позиція $P_2$ перевіряє відповідність параметрів циклу на черговій ітерації (в ході імплементації моделі буде реалізовано алгоритмічно). Власне новаційність поданої моделі обходу мережевого графа полягає у циклічності встановлення зв'язків між подіями та виконанні розрахунку раннього терміну виконання подій в межах поточної ітерації. Перехід $t_2$ реалізує процедуру

входження у цикл та визначення поточного ребра (роботи) $i - j$. Позиція $P_3$ перевіряє умову чи ребро $i - j$ не належить множині $\{t_{i-j}\}$ (тобто мережевим графіком робота $i - j$ не передбачена): $i - j \notin \{t_{i-j}\}$. Перехід $t_3$ відповідає за визначення поточного значення події $j$, адже на кожній ітерації циклу значення індексу наступної події $j$ буде змінюватись. В позиції $P_4$ проводиться перевірка, чи значення $j$ менше або однакове з кількістю подій мережевої моделі $n$ $(j \le n)$. Якщо нерівність $j \le n$ виконується, здійснюється перехід $t_4$, де значення індексу $j$ збільшується на 1 $(j{+}{+})$. Після чого процедура обходу мережевого графа повертається у позицію $P_2$. Позиція $P_5$ перевіряє зворотну умову, коли значення $j$ більше за кількість подій мережевої моделі $n$. За умови виконання нерівності $j > n$, здійснюється перехід $t_5$, де індекс $i$ збільшується на 1 $(i{+}{+})$. В позиції $P_6$ здійснюється перевірка нерівності індексу попередньої події та кількості подій мережевої моделі $n: i \ne n$ (не досягнуто граничного значення циклу). Якщо нерівність виконується – спрацьовує перехід $t_6$, який відповідає за встановлення нового значення індексу $j = i+1$. Після чого здійснюється циклічне повернення до перевірки умови $P_2$. Позиція $P_7$ перевіряє рівність індексу попередньої події $i$ та кількості подій мережевої моделі $n$. Якщо значення індексу $i$ досягає $n$ (досягнуто граничного значення циклу), то виконується перехід $t_7$, із виводом результатів розрахунку та подальшим завершенням обходу мережевого графа у позиції $P_8$.

Повернемось до розгалуження моделі після виконання переходу $t_2$ (вхід у цикл та перевірка поточного значення $i$ - $j$). В позиції $P_9$ відбувається перевірка умови, чи поточна робота $i$ - $j$ належить множині $\{ti\text{-}j\}$ (вхідними даними мережевої моделі передбачена робота $i - j$). У разі наявності відповідної роботи у заданій множині робіт $i - j \in \{t_{i-j}\}$ виконується перехід $t_8$, завданням якого є визначення раннього терміну виконання події на поточному етапі обходу мережевого графа для події j за залежністю:

$$T_{f(j)} = T_{f(i)} + t_{(i \to j)}, \tag{2.30}$$

де $T_{f(i)}$ – ранній термін виконання попередньої події; $t_{(i\to j)}$ – обсяг роботи, яка передує події $j$ (враховує складність та обсяги робіт у Story Point).

Далі у позиції $P_{10}$ здійснюється перевірка, чи значення розрахованого раннього терміну $T_{f(j)}$ на поточній ітерації циклу не міститься у множині ранніх термінів виконання подій: $T_{f(j)} \notin \{T_{fj}\}$. Якщо поточне значення раннього терміну $T_{f(j)}$ не міститься у множині, це означає, що значення отримано вперше. Після чого в переході $t_9$ відбувається визначення попередньої події $T_i$ для поточної події $T_j$: $T_{prev(j)} = T_i$, (фіксується номер попередньої події для $T_j$). В позиції $P_{11}$ відбувається перевірка факту запису (наявності) $T_{prev(j)}$ до поточної події $T_j$ (факт запису = true). В переході $t_{10}$ відбувається запис зв'язку поточна подія $T_j$ – попередня подія $T_{prev(j)}$, до *HashMap* $<T_j, T_{prev(j)}>$. Позиція $P_{12}$ перевіряє факт запису зв’язку $T_j - T_{prev(j)}$ до *HashMap* $< T_j, T_{prev(j)}>$ (факт запису = true). Визначене унікальне значення раннього терміну виконання подій $T_{f(j)}$ на поточній ітерації циклу записується до множини ранніх термінів *HashSet* $\{T_{f(j)}\}$ в події $t_{11}$. В позиції $P_{13}$ перевіряється чи значення $T_{f(j)}$ успішно записано до множини *HashSet* $\{T_{f(j)}\}$ (факт запису = true). За виконання умови $P_{13}$ виконується перехід до $t_3$ та початок нової ітерації циклу.

Позиція $P_{14}$ перевіряє наявність визначеного у переході $t_8$ значення раннього терміну виконання подій $T_{f(j)}$ у множині ранніх термінів $\{T_{fj}\}$ після чергової ітерації розрахунків (наявність $T_{f(j)}$ у множині $\{T_{fj}\}$ можлива, якщо події $T_j$ передують декілька подій $T_i$, а на попередніх ітераціях циклу відбувалось визначення показника їх раннього терміну виконання). У випадку виконання умови $P_{14}$ $T_{f(j)} \in \{T_{fj}\}$ виконується перехід $t_{12}$, де визначається максимальне значення серед терміну $T_{f(j)}$ поточної ітерації та попередньо записаного значення $T_{f(j)}$ до множини $\{T_{fj}\}$ (за результатами попередніх ітерацій). На цьому етапі необхідно передбачити можливість коректного розрахунку часу раннього початку для тих подій, яким передують декілька робіт з врахуванням вимоги $T_{f(j)} \to \max$, використовуючи вираз:

$$T_{f(j)} = \max \begin{cases} T_{f(k)} + t_{(k \to j)} \\ \cdots\cdots\cdots \\ T_{f(m)} + t_{(m \to j)} \end{cases}, \tag{2.31}$$

де *k, m* – це індекси подій, що передують події *j*.

У позиції $P_{15}$ відбувається перевірка нерівності $\{T_{f(j)}\} \geq T_{f(j)}$, де встановлюється чи значення раннього терміну виконання подій, яке записано до множини $\{T_{f(j)}\}$ на попередніх ітераціях більше або рівне поточному розрахованому значенню раннього терміну $T_{f(j)}$. Якщо нерівність виконується, то перезапису збереженого у множині значення поточним не відбувається (записане значення більше за поточно розраховане), а система переходить на нову ітерацію циклу $t_3$. В позиції $P_{16}$ відбувається зворотна перевірка нерівності $\{T_{f(j)}\} < T_{f(j)}$. Якщо поточне розраховане значення раннього терміну $T_{f(j)}$ більше за раніше записане значення до множини $\{T_{f(j)}\}$, запускається процедура перезапису більшого значення до $HashSet\{T_{f(j)}\}$ шляхом виконання переходу $t_9$ із подальшим проходження черговості описаних етапів.

Представлена на рисунку 2.10 імітаційна модель обходу мережевого графа дала змогу описати процедуру визначення раннього терміну виконання подій $T_{f(j)}$ із урахуванням усіх обмежень у вигляді алгоритму. Імітаційне моделювання процесу обходу мережевого графа дозволило розробити унікальний алгоритм та адаптувати його під методологію гнучкого управління проєктними роботами. Термін раннього виконання подій для останньої події мережевої моделі *n* є терміном виконання усього комплексу робіт спринта.

Далі проведено імітаційне моделювання процесу обходу мережевого графа з метою визначення пізнього терміну виконання робіт $T_l$. В основу цієї моделі також закладено принцип циклічного перебору зв'язків $i-j$, проте відмінністю є зворотній порядок обходу (від останньої до першої події). Визначення пізнього терміну виконання подій $T_{l(i)}$ реалізується з метою контролю термінів, до яких події мережевої моделі мають бути повністю виконані задля уникнення випадків

збільшення часу на реалізацію усього проєкту. Розглянемо модель цього процесу у вигляді мережі Петрі на рисунку 2.11.

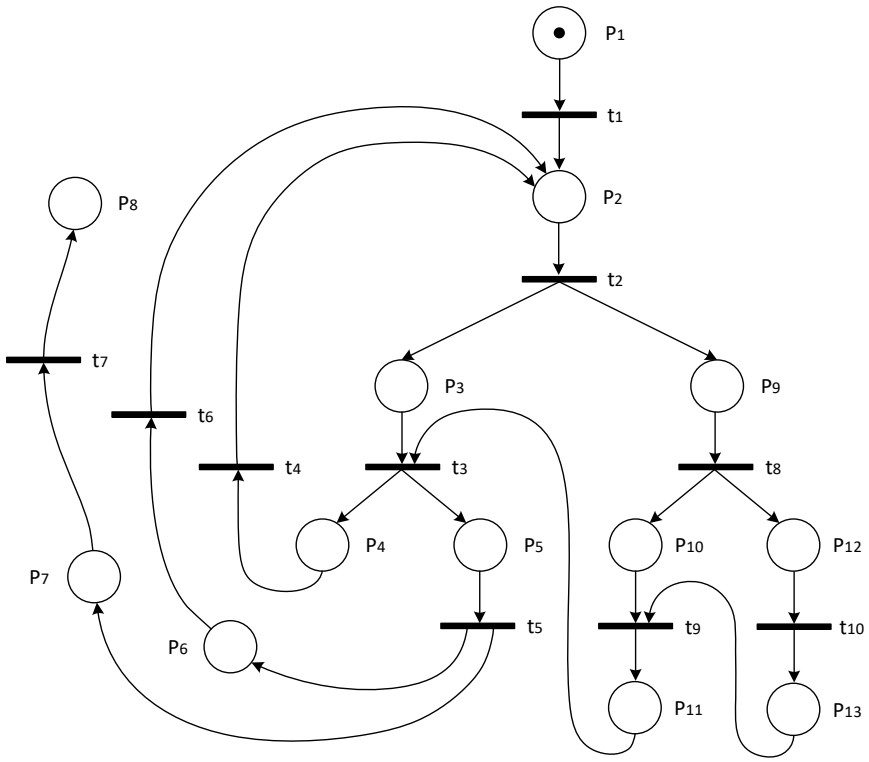



Рисунок 2.11 – Імітаційна модель процесу обходу мережевого графа з метою визначення пізнього часу виконання робіт $T_l$

Описаний імітаційною моделлю процес розрахунку пізнього терміну виконання події починається із позиції $P_1$, яка перевіряє введення даних: індексу попередньої події $i=0;$ індексу поточної події $j = n;$ обсягу подій мережевої моделі $n;$ множини робіт мережевої моделі $\{t_{i-j}\}$; множини ранніх термінів виконання подій $\{T_{fj}\}$ (розраховано за результатами обходу графа за попередньою моделлю). За умови виконання $P_1$ здійснюється перехід $t_1$ – підготовка параметрів циклічного обходу графа, де пізній час виконання для останньої події рівний ранньому часу виконання цієї події $T_{l(n)}=T_{f(n)},$ а значення індексу $i = j–1$. Позиція $P_2$ перевіряє відповідність параметрів циклу на черговій ітерації. Перехід $t_2$ реалізує процедуру входження у цикл та визначення поточного ребра (роботи) $i – j$. Циклічна процедура обходу мережевого графа організована за прикладом попередньої моделі, відмінністю є зворотній шлях обходу графа (з кінця до початку). Позиція $P_3$ перевіряє умову чи ребро $i – j$ не належить множині $\{t_{i-j}\}$: $i – j \notin \{t_{i-j}\}$. Перехід $t_3$

відповідає за визначення поточного значення попередньої події *i*, адже на кожній ітерації циклу значення індексу попередньої події *i* буде зменшуватись. В позиції $P_4$ проводиться перевірка, чи значення *i* більше або рівне за 1 ($i \geq 1$). Якщо нерівність $i \geq 1$ виконується, здійснюється перехід $t_4$, де значення індексу *i* зменшується на 1 (*i*—). Після чого процедура обходу мережевого графа повертається у позицію $P_2$. Позиція $P_5$ перевіряє зворотну умову, коли значення $i < 1$. За умови виконання нерівності $i < 1$, здійснюється перехід $t_5$, де індекс *j* зменшується на 1 (*j*—). В позиції $P_6$ здійснюється перевірка чи значення індексу поточної події *j* не досягло 1: $j \neq 1$ (не досягнуто граничного значення циклу). Якщо нерівність виконується, спрацьовує перехід $t_6$, який відповідає за встановлення нового значення індексу $i = j-1$. Після цього здійснюється циклічне повернення до перевірки умови в позиції $P_2$. Позиція $P_7$ перевіряє рівність індексу поточної події $j = 1$. Якщо значення індексу *j* досягає 1 (досягнуто граничного значення циклу), то виконується перехід $t_7$, із виводом результатів розрахунку та подальшим завершенням обходу мережевого графа у позиції $P_8$.

Перейдемо до розгалуження моделі після виконання переходу $t_2$ (вхід у цикл та перевірка поточного значення $i - j$). В позиції $P_9$ відбувається перевірка умови, чи поточна робота $i - j$ належить множині $\{t_{i\text{-}j}\}$. У разі наявності відповідної роботи у заданій множині робіт $i - j \in \{t_{i\text{-}j}\}$ виконується перехід $t_8$, завданням якого є визначення пізнього терміну з виразу:

$$T_{l(i)} = T_{l(j)} - t_{(i \to j)}, \tag{2.32}$$

де $T_{l(j)}$ – пізній термін виконання наступної (поточної) події; $t_{(i \to j)}$ – обсяг роботи, яка виконується після події *i* та перед подією *j* (враховує обмеження у Story Point).

Далі у позиції $P_{10}$ здійснюється перевірка, чи значення розрахованого пізнього терміну $T_{l(i)}$ на поточній ітерації циклу не міститься у множині пізніх термінів виконання подій: $T_{l(i)} \notin \{T_{l(i)}\}$. Якщо поточне значення пізнього терміну $T_{l(i)}$ не міститься у множині, це означає, що розраховане значення отримано вперше. Після чого в переході $t_9$ відбувається запис значення пізнього терміну виконання цієї події

$T_{l(i)}$ до множини $HashSet\{T_{li}\}$. В позиції $P_{11}$ відбувається перевірка факту запису (наявності) $T_{l(i)}$ до $HashSet\{T_{li}\}$ (факт запису = true), після чого відбувається перехід на нову ітерацію циклу в переході $t_3$. Позиція $P_{12}$ перевіряє наявність визначеного у переході $t_8$ значення пізнього терміну виконання подій $T_{l(i)}$ у множині пізніх термінів $\{T_{l(i)}\}$ після чергової ітерації розрахунків. У випадку виконання умови $T_{l(i)} \in \{T_{l(i)}\}$ виконується перехід $t_{10}$, де визначається мінімальне значення серед терміну $T_{l(i)}$ поточної ітерації та попередньо записаного значення $T_{l(i)}$ до множини $\{T_{l(i)}\}$, використовуючи вираз:

$$T_{l(i)} = \min \begin{cases} T_{l(k)} - t_{(i \to k)} \\ \cdots\cdots\cdots \\ T_{l(m)} - t_{(i \to m)} \end{cases}, \tag{2.33}$$

де $k$, $m$ – це індекси похідних подій, від події $i$.

В позиції $P_{13}$ відбувається перевірка факту визначення мінімального значення $T_{l(i)}$ на поточній ітерації циклу, після чого виконується його перезапис до множини $HashSet\{T_{li}\}$ шляхом повернення до переходу $t_9$.

Пізній термін $T_l$ для останньої події мережевої моделі рівний часу її раннього виконання події $T_f$ :

$$T_{l(n\ ост.)} = T_{f(n\ ост.)}. \tag{2.34}$$

Значення $T_l$ для першої події мережевої моделі за умови вірного виконання попередніх розрахунків буде дорівнювати часу раннього її виконання $T_f$:

$$T_{l(n\ поч.)} = T_{f(n\ поч.)}. \tag{2.35}$$

Для усіх подій мережевої моделі, крім першої та останньої, має виконуватись нерівність:

$$T_{l(n)} \geq T_{f(n)}. \tag{2.36}$$

Представлена на рисунку 2.11 імітаційна модель обходу мережевого графа дала змогу описати процедуру ітераційного визначення пізнього терміну виконання подій $T_{l(i)}$ із урахуванням усіх обмежень у вигляді алгоритму.

Реалізовані імітаційні моделі та побудовані на їх основі алгоритми обходу мережевого графа для розрахунку раннього та пізнього термінів виконання подій дають змогу оцінити максимальну тривалість робіт з розроблення спеціалізованого програмного забезпечення зважаючи на наявні ресурси, які необхідні для успішного виконання спринта. Отримані алгоритми на основі імітаційних моделей надають можливість автоматизувати визначення раннього та пізнього термінів виконання робіт, а відтак реалізувати процедуру оперативного перепланування тривалості проєктних робіт в динамічному оточенні (в умовах постійних змін).

Унікальність імітаційних моделей обходу мережевого графа полягає у циклічності процедури встановлення зв'язку між попередньою та поточною подією $i - j$. При обході мережевого графа для визначення раннього терміну за умови фіксованого значення індексу попередньої події $i$ здійснюється ітераційне збільшення індексу поточної події $j$ до моменту, коли значення $j$ буде більше за кількість подій мережевої моделі $n$. На цьому етапі циклічний перебір завершується, індекс попередньої події $i$ збільшується на одиницю та реалізується процедура повторного входу у цикл із значенням $j = i+1$. Процедура повторного виконання циклу для обходу мережевого графа виконується до моменту коли значення індексу попередньої події і досягне значення кількості подій мережевої моделі $n$. При обході мережевого графа для визначення пізнього терміну, процедура має аналогічний підхід, проте у зворотньому порядку (від останньої до першої події). За цих умов значення індексу $j$ на початковому етапі рівний $n$, а індекс $i$ ітераційно зменшується на 1. Застосування цього підходу дозволить почергово перебирати усі ребра (роботи) мережевої моделі із подальшим визначенням необхідних показників.

На основі реалізованих імітаційних моделей обходу мережевого графа для розрахунку раннього та пізнього термінів виконання подій, вдалося оцінити максимальну тривалість робіт з розроблення спеціалізованого програмного забезпечення, враховуючи наявні ресурси для успішного виконання спринта. Унікальність представлених моделей полягає в циклічності процедури встановлення зв'язку між попередньою та поточною подією, що дозволяє почергово перебирати всі ребра мережевої моделі. Отримані моделі створюють передумови для автоматизації визначення раннього та пізнього термінів виконання робіт і, як наслідок, реалізації процедури оперативного перепланування тривалості проєктних робіт в динамічному оточенні постійних змін.

### 2.4. Висновки

За результатами розділу можна зробити такі висновки:

1. Моделюванням процесу розробки програмних систем із використанням понятійного апарату теорії множин отримано математичну модель їх життєвого циклу, яка дає змогу охарактеризувати обсяги робіт на окремих етапах створення програмного продукту, встановити взаємозв'язки і взаємозалежності між ними та сформулювати фундаментальні принципи автоматизації процедури підтримки прийняття управлінських рішень на різних етапах створення програмних систем, зокрема, безпеко-орієнтованого спрямування;

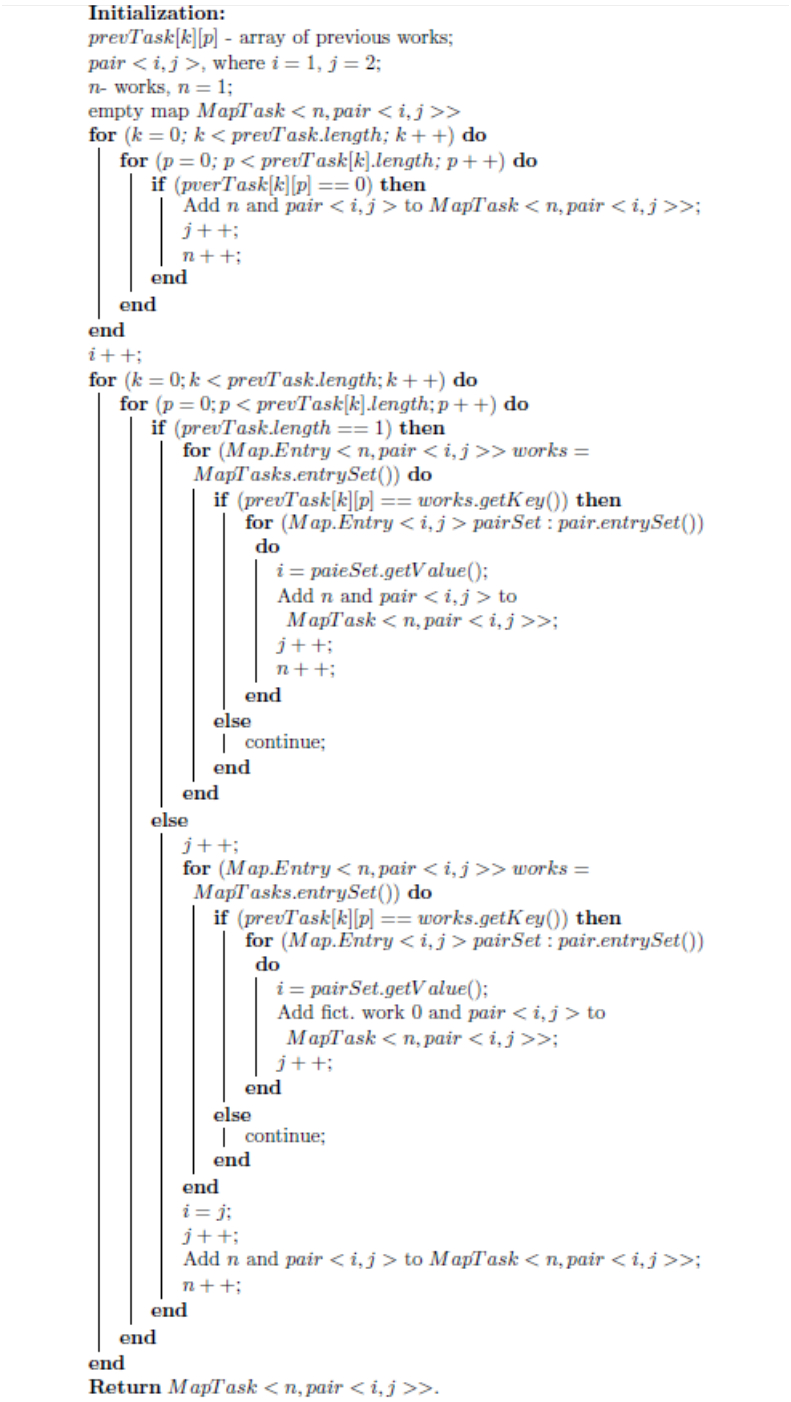
2. Розроблено концептуальну модель процесу управління життєвим циклом розроблення спеціалізованого програмного забезпечення (безпеко-орієнтованих сервісів), котра адаптована під специфіку роботи Державної служби України із надзвичайних ситуацій та корелює із принципами гнучкої методології управління ІТ-проєктів.

3. Шляхом імітаційного моделювання з використанням понятійного апарату мереж Петрі отримано уніфіковану модель процесів обходу мережевих графів, яка полягає у циклічності процедури встановлення зв'язку між попередньою і поточною подіями та уможливлює її адаптивне застосування не залежно від складності та конфігурації мережевих графів.

# РОЗДІЛ ІІІ. РОЗРОБЛЕННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПІДТРИМКИ РІШЕНЬ

## 3.1. Алгоритми обходу мережевого графа, які покладені в основу системи підтримки рішень

Першим етапом для визначення обсягів проєктних робіт необхідно побудувати мережевий граф. Для цього нумеруємо початкову подію *j*=1, попередня подія *(i = 0)* – відсутня. Від неї проводимо вектор, який позначатиме першу користувацьку історію чи функцію та її вагу у s.p. Якщо є декілька робіт, які можуть виконуватись паралельно, будується необхідна кількість векторів. Оскільки кожна користувацька історія має завершуватись кінцевою подією, позначаєм подію на графіку і збільшуємо її порядковий номер на $j = i + 1$ (робимо так для всіх паралельних робіт, при цьому $j$ збільшуємо на 1). Якщо потрібно побудувати користувацьку історію, виконання якої можливе тільки після попередньої, шукаємо на графіку кінцеву подію роботи, яка нас цікавить та від неї проводимо вектор потрібної користувацької історії. Якщо ж виконання користувацької історії залежить від завершених декількох попередніх, цю задачу вирішуємо із використанням фіктивної роботи. Від кінця кожної попередньої користувацької історії проводимо вектор, який означає фіктивну роботу вага якої 0 s.p. та завершуємо їх у одній кінцевій події, яка служитиме початком потрібної нам нової користувацької історії. Таким чином будуємо графік мережевого планування для тої кількості користувацьких історій та функцій, які нас цікавлять (в межах обсягу спринта). Розроблений нами алгоритм побудови мережевого графа (мережевої моделі) визначеного обсягу робіт на спринт наведений на рисунку 3.1.



**Initialization:**
$prevTask[k][p]$ - array of previous works;
$pair < i, j >$, where $i = 1$, $j = 2$;
$n$- works, $n = 1$;
empty map $MapTask < n, pair < i, j >>$
**for** $(k = 0;\ k < prevTask.length;\ k++)$ **do**
  **for** $(p = 0;\ p < prevTask[k].length;\ p++)$ **do**
    **if** $(pverTask[k][p] == 0)$ **then**
      Add $n$ and $pair < i, j >$ to $MapTask < n, pair < i, j >>$;
      $j++$;
      $n++$;
    **end**
  **end**
**end**
$i++$;
**for** $(k = 0; k < prevTask.length; k++)$ **do**
  **for** $(p = 0; p < prevTask[k].length; p++)$ **do**
    **if** $(prevTask.length == 1)$ **then**
      **for** $(Map.Entry < n, pair < i, j >> works =$
       $MapTasks.entrySet())$ **do**
        **if** $(prevTask[k][p] == works.getKey())$ **then**
          **for** $(Map.Entry < i, j > pairSet : pair.entrySet())$
           **do**
            $i = paieSet.getValue()$;
            Add $n$ and $pair < i, j >$ to
             $MapTask < n, pair < i, j >>$;
            $j++$;
            $n++$;
          **end**
        **else**
          continue;
        **end**
      **end**
    **else**
      $j++$;
      **for** $(Map.Entry < n, pair < i, j >> works =$
       $MapTasks.entrySet())$ **do**
        **if** $(prevTask[k][p] == works.getKey())$ **then**
          **for** $(Map.Entry < i, j > pairSet : pair.entrySet())$
           **do**
            $i = pairSet.getValue()$;
            Add fict. work 0 and $pair < i, j >$ to
             $MapTask < n, pair < i, j >>$;
            $j++$;
          **end**
        **else**
          continue;
        **end**
      **end**
      $i = j$;
      $j++$;
      Add $n$ and $pair < i, j >$ to $MapTask < n, pair < i, j >>$;
      $n++$;
    **end**
  **end**
**end**
**Return** $MapTask < n, pair < i, j >>$.

Рисунок 3.1 – Алгоритм для побудови мережевого графа

Слід нагадати, що основними елементами мережевої моделі є її події та роботи, які оцінюються у Story Point. Такий різновид оцінки дозволяє враховувати часові та ресурсні обмеження процесів розробки специфічного програмного забезпечення.

До основних параметрів мережевої моделі для часової оцінки процесів розробки спеціалізованого програмного забезпечення на стадії планування,

віднесено: ранній термін виконання подій $T_f$, пізній термін виконання подій $T_l$, резерви виконання подій $R_{подій}$, резерви виконання робіт $R_{робіт}$. Алгоритм визначення означених параметрів, який закладено в роботу інформаційної системи підтримки прийняття рішень, розглянемо далі.

На першому етапі реалізується визначення раннього терміну виконання подій. На рисунку 3.2 зображений алгоритм визначення раннього часу виконання події, враховуючи розроблену імітаційну модель, описану у другому розділі.

**Initialization:**
$i$ - index of the previous event, $i = 0$;
$j$ - index of the next event, $j = 1$;
$n$ - the volume of network model events;
$\{t_{i \to j}\}$- set of network model works;
$\forall j = 1$ then $T_{e(1)} = 0$ ;
$i++$;
**for** $(j = i + 1;\ j \le n;\ j++)$ **do**
  **if** $(i \to j \in \{t_{i \to j}\})$ **then**
    $T_{e(j)} = T_{e(i)} + \{t_{i \to j}\}$;
    **if** $(T_{e(j)} \in \{T_{e_j}\})$ **then**
      $max(\{T_{e_j}\}; \{T_{e(j)}\})$;
      **if** $(\{T_{e_j}\} < T_{e(j)})$ **then**
        $T_{prev(j)} \leftarrow T_i$;
        $HashMap < T_j, T_{prev(j)} > \leftarrow < T_j, T_{prev(j)} >$;
        $HashSet\{T_{e_j}\} \leftarrow T_{e(j)} >$;
      **else**
        continue $for$;
      **end**
    **else**
      $T_{prev(j)} = T_i$;
      $HashMap < T_j, T_{prev(j)} > \leftarrow < T_j, T_{prev(j)} >$;
      $HashSet\{T_{e_j}\} \leftarrow T_{e(j)}$;
      continue $for$;
    **end**
  **else**
    continue $for$;
  **end**
**end**
$i++$;
**if** $(i \neq n)$ **then**
  start $for$;
**end**
**Return:**
$\{T_{e_j}\}$ - the set of early term execution of $n$ event of the network model;
$HashMap < T_j, T_{prev(j)} >$ - the set of previous events for the next events in the format $< Key, Value >$.

Рисунок 3.2 – Алгоритм для визначення раннього часу виконання події

Далі з метою контролю термінів, до яких події мережевої моделі мають бути повністю виконані, задля уникнення випадків збільшення часу на реалізацію усього

проєкту, необхідно визначити пізній термін виконання подій $T_{l(i)}$, за таким алгоритмом:

**Initialization:**
$i$ - index of the previous event, $i = 0$;
$j$ - index of the next event, $j = n$;
$n$ - the volume of network model events;
$\{t_{i \to j}\}$- set of network model works;
$\{T_{e_j}\}$- set of early time execution events;
$\forall j = n$ then $T_{l(n)} = T_{e(n)}$ ;
**for** $(i = j - 1;\ i \geq 1;\ i - -)$ **do**
  **if** $(i \to j \in \{t_{i \to j}\})$ **then**
    $T_{l(i)} = T_{l(j)} - \{t_{i \to j}\}$;
    **if** $(T_{l(i)} \in \{T_{l_t}\})$ **then**
      $min(\{T_{l_t}\}; \{T_{l(i)}\})$;
      **if** $(\{T_{l_t}\} < T_{l(i)})$ **then**
        $HashSet\{T_{l_t}\} \leftarrow T_{l(i)}$;
      **else**
        continue $for$;
      **end**
    **else**
      $HashSet\{T_{l_t}\} \leftarrow T_{l(i)}$;
      continue $for$;
    **end**
  **else**
    continue $for$;
  **end**
**end**
$j - -$;
**if** $(j \neq 1)$ **then**
  start $for$;
**end**
**Return:**
$\{T_{l_t}\}$ - the set of late term execution of $n$ event of the network model;

Рисунок 3.3 – Алгоритм для визначення пізнього часу виконання події

Результати проведених розрахунків дозволяють оцінити максимальні обсяги робіт зважаючи на наявні ресурси, які необхідні для успішного виконання спринта. Максимальні обсяги робіт в мережевій моделі відображає так званий критичний шлях. Визначення критичного шляху організовується мережею робіт $t_{(i \to j)}$ від чергової пізньої події до події, що їй передує (передньої події зазначеної у відповідному секторі вершини мережевого графа). Побудова критичного шляху починається з кінцевої події $T_n$ та завершується початковою подією $T_1$:

$$T_n \to [t_{j \to n}] \to T_j \to [t_{k \to j}] \to T_k \to \ldots \to T_m \to [t_{i \to m}] \to T_i \to [t_{1 \to i}] \to T_1, \qquad (3.1)$$

де $T_n$ – остання подія мережевої моделі (перша для критичного шляху);

$T_j$ – пізня проміжна подія мережевої моделі;

$T_i$ – рання проміжна подія мережевої моделі;

$T_k$ , $T_m$ – проміжні події мережевої моделі;

$T_1$ – перша подія мережевої моделі (остання для критичного шляху).

Алгоритмічно процедура визначення критичного шляху за результатами обходу графа мережевої моделі представлена на рисунку 3.4. Вхідними даними для виконання цієї процедури є результати виконання попередніх алгоритмів.

**Initialization:**
$j$ - index of the event, $j = n$;
$k$ - temporary variable;
$HashMap < T_j, T_{prev} >$ - the set of previous events for the next events $T_j$ in the format $< Key, Value >$.
$Stack.add(T_j)$;
**while** $(j \geq 1)$ **do**
    $Stack.add(k)$;
    $j - -$;
**end**
**Return:**
$Stack$.

Рисунок 3.4 – Алгоритм обходу графа мережевої моделі для визначення критичного шляху

Особливість критичного шляху виконання робіт полягає в тому, що в його межах не міститься жодного ресурсного резерву на досягнення подій (ранній та пізній терміни виконання усіх подій на критичному шляху тотожні). Це свідчить про те, що будь-яка затримка на виконання подій мережевої моделі, які лежать в межах критичного шляху, стимулюватиме до неповного виконання визначеного обсягу робіт на спринт або перевищення допустимого часу їх реалізації.

Проте інші роботи, що входять до складу мережевої моделі та не належать до критичного шляху, можуть мати певні резерви на досягнення подій та виконання робіт. Значення цих резервів дозволятиме контролювати межі критичного часу початку та завершення робіт, що не лежать в межах критичного шляху. Розрахунок означених ресурсних резервів проводиться з використанням таких виразів:

$$R_{події}(j) = T_{l(j)} - T_{f(j)}; \tag{3.2}$$

$$R^{п}_{роботи}(t_{i\to j}) = T_{l(j)} - T_{f(i)} - t_{(i\to j)}; \tag{3.3}$$

$$R^{в}_{роботи}(t_{i\to j}) = T_{f(j)} - T_{f(i)} - t_{(i\to j)}; \tag{3.4}$$

$$R^{н}_{роботи}(t_{i\to j}) = T_{f(j)} - T_{l(i)} - t_{(i\to j)}, \tag{3.5}$$

де $R_{події}(j)$ – резерв часу події;

$R^{н}_{роботи}(t_{i\to j})$ – незалежний резерв роботи;

$R^{п}_{роботи}(t_{i\to j})$ – повний резерв роботи;

$R^{в}_{роботи}(t_{i\to j})$ – вільний резерв роботи;

$T_{l(i)}$ – пізній термін попередньої події;

$T_{l(j)}$ – пізній термін наступної події;

$T_{f(i)}$ – ранній термін попередньої події;

$T_{f(j)}$ – ранній термін наступної події;

$t$ – обсяг роботи;

$j$ – наступна подія мережевої моделі;

$i$ – попередня подія мережевої моделі.

Резерв часу події $R_{події}(j)$ характеризує максимально допустимий період часу, на який можливо затримати виконання події $n$ без збільшення критичного шляху та ресурсного обмеження на спринт. Повний резерв роботи $R^{п}_{роботи}(t_{i\to j})$ надає характеристику на який час можливо перенести початок виконання роботи $t_{i\to j}$ або збільшити її тривалість в межах наявних ресурсів та без збільшення загального часу спринта (довжини критичного шляху). Вільний резерв роботи $R^{в}_{роботи}(t_{i\to j})$ це час на який можливо перенести початок виконання роботи $t_{i\to j}$ або розтягнути її тривалість, не порушуючи ранніх термінів виконання подій мережевої моделі. Незалежний резерв роботи $R^{н}_{роботи}(t_{i\to j})$ характеризує час затримки початку виконання роботи $t_{i\to j}$, не збільшуючи загального терміну виконання комплексу робіт спринта та не затримуючи жодну з інших робіт мережевої моделі. Алгоритм

обходу графа мережевої моделі для визначення означених резервів представлено на рисунку 3.5.

**Initialization:**
$i$ - index of the previous event, $i = 0$;
$j$ - index of the next event, $j = n$;
$n$ - the volume of network model events;
$\{t_{i \to j}\}$- set of network model works;
$\{T_{e_j}\}$- set of the early terms executed events;
$\{T_{l_j}\}$- set of the late terms executed events;
$Stack$ - the critical path of network model;
**while** $(j \geq 1)$ **do**
  $R_{e(j)} = T_{l(j)} - T_{e(j)}$;
  **if** $(R_{(j)} == 0)$ **then**
    $j--$;
    continue *while*;
  **else**
    $HashSet\{R_j\} \to R_{(j)}$;
    $j--$;
    continue *while*;
  **end**
**end**
$i \to n$;
**for** $(i = j - 1; i \geq 1; i--)$ **do**
  **if** $(i \to j \in \{t_{i \to j}\})$ **then**
    **if** $(T_i \notin Stack)$ **then**
      $R_{pw}(t_{(i \to j)}) = T_{l_{(j)}} - T_{e_{(i)}} - t_{(i \to j)}$;
      $R_{fw}(t_{(i \to j)}) = T_{e_{(j)}} - T_{e_{(i)}} - t_{(i \to j)}$;
      $R_{dw}(t_{(i \to j)}) = T_{e_{(j)}} - T_{l_{(i)}} - t_{(i \to j)}$;
      $HashSet\{R_{pw}(t_{(i \to j)})\} \to R_{pw}(t_{(i \to j)})$;
      $HashSet\{R_{fw}(t_{(i \to j)})\} \to R_{fw}(t_{(i \to j)})$;
      $HashSet\{R_{dw}(t_{(i \to j)})\} \to R_{dw}(t_{(i \to j)})$;
      continue *for*;
    **else**
      continue *for*;
    **end**
  **else**
    continue *for*;
  **end**
**end**
$j--$;
**if** $(j \neq 1)$ **then**
  start *for*;
**else**
  **Return:**
  $\{R_{e(j)}\}, \{R_{pw}(t_{(i \to j)})\}, \{R_{fw}(t_{(i \to j)})\}, \{R_{dw}(t_{(i \to j)})\}$.
**end**

Рисунок 3.5 – Алгоритм обходу графа мережевої моделі для визначення часових резервів

## 3.2. Архітектура та принципи побудови інформаційної технології управління ЖЦПЗ

Реалізація інформаційної системи підтримки прийняття рішень в управлінні життєвим циклом розроблення спеціалізованого програмного забезпечення представлена у форматі web-застосунку. Дана система розроблена із використанням мови програмування Java та фреймворка Spring Boot, для розробки

зовнішньої частини сайту використано html, CSS (Bootstrap 5), JavaScript та Thymeleaf (серверний механізм Java-шаблонів призначений для обробки HTML, XML, JavaScript, CSS).

Розглянемо концептуальну модель інформаційної системи підтримки прийняття рішень в управлінні життєвим циклом спеціалізованого програмного забезпечення (далі – ІСППР в УЖЦ СПЗ) шляхом візуалізації архітектури клієнтської частини застосунку (рисунок 3.6.). Архітектура цієї частини застосунку відповідає за реакцію на дії користувача та взаємодію із сервером. У ролі потенційного «клієнта» ми розглядаємо проєкт менеджера (PM), власника продукту (PO) або ж скрам майстра (SM), залежно від обсягу проєкту, складу команди тощо. «Клієнт» вводить дані із користувацькими історіями, що бере команда в розробку на спринт у систему, згідно їх пріоритетності та можливостей команди. Далі ці дані зберігаються у беклозі спринта, а також використовуються для побудови мережевого графа планування. Після введення, обробки та зберігання інформації активується алгоритмічний блок обчислень та блок візуалізації мережевого графа планування, які представлені на користувацькому інтерфейсі веб-застосунку. Оскільки користувач має можливість редагувати дані, то між блоками введення та редагування даних існує двосторонній зв'язок.

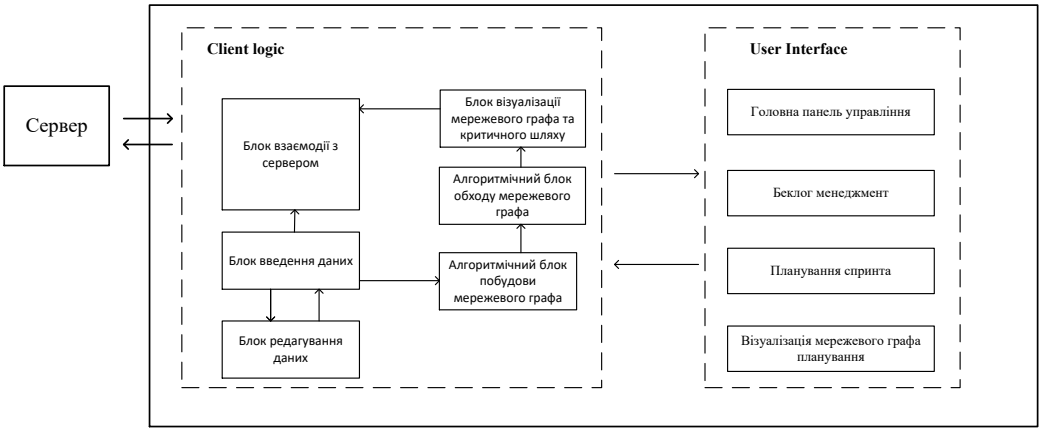



Рисунок 3.6 - Архітектура клієнтської частини web-застосунку

Основне призначення роботи серверної частини (рисунок 3.7) в автоматичному режимі – це опрацювання запитів, що надходять з клієнтської частини та зворотне надсилання результатів їх оброблення через блок взаємодії з клієнтом.

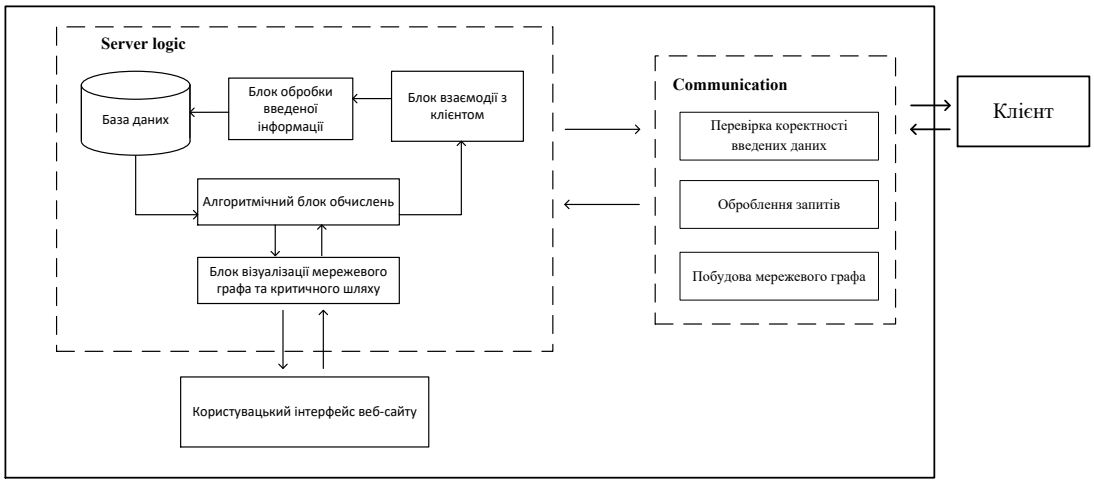



Рисунок 3.7 - Архітектура серверної частини системи

На рисунку 3.8 зображена UML діаграма основних класів даної системи.

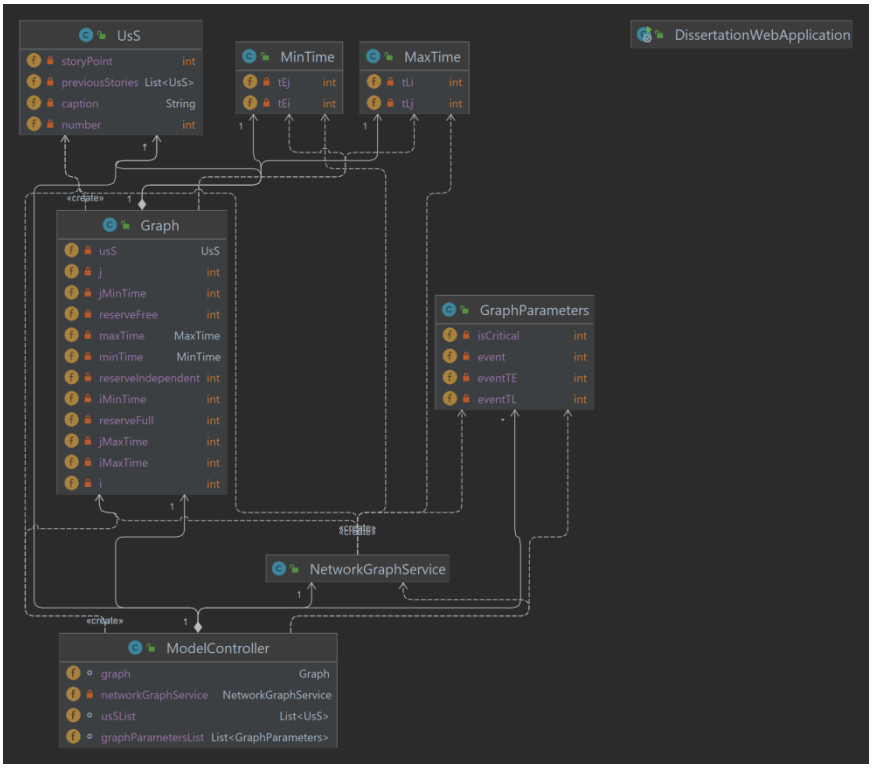



Рисунок 3.8 – UML діаграма класів програмної системи

З даної діаграми можна спостерігати, що для побудови мережевого графа було створено 4 сутності, зокрема Graph, UsS, MinTime та MaxTime.

Для зручності у класі Graph було реалізовано метод із побудови мережевого графа, а інші методи були реалізовані у класі NetworkGraphService та передані ModelController.

Якщо ж говорити про користувацький інтерфейс та функціонал, то головна сторінка програми має такий вигляд (рисунок 3.9):

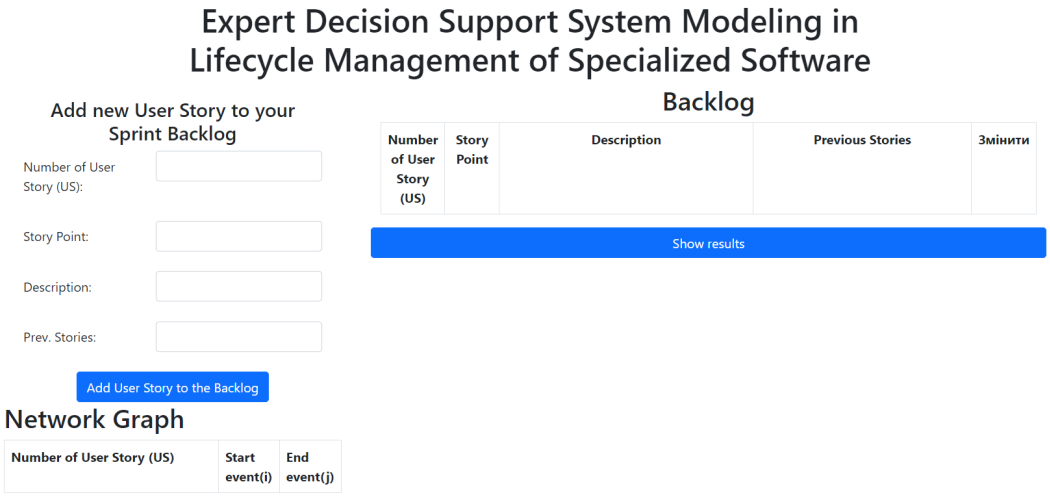



Рисунок 3.9 – Головна сторінка web-застосунку ІСППР в УЖЦ СПЗ

На цій сторінці реалізований функціонал додавання користувацької історії у беклог, перегляд та редагування беклогу. Окрім цього сформована таблиця із побудованим мережевим графом (на основі розробленого алгоритму побудови мережевого графа), де наявними є номер користувацької історії а також початкові та кінцеві події виконання. Приклад реалізації функціоналу головної сторінки зображений на рисунку 3.10.

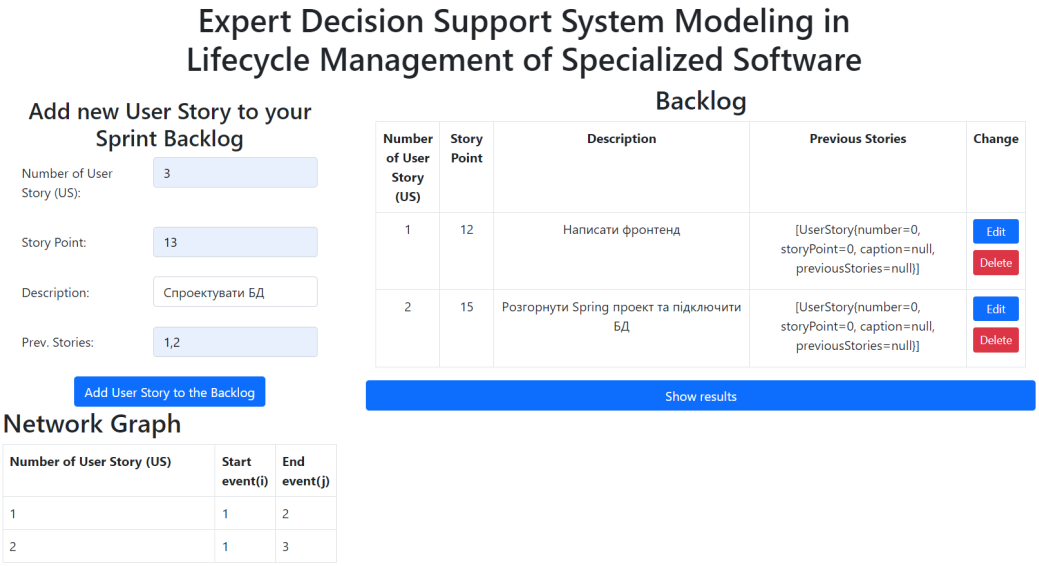



Рисунок 3.10 – Приклад реалізації функціоналу

При натисканні на кнопку «Show results» появиться нове вікно, в якому будуть обчислюватись результати роботи розроблених алгоритмів обходу мережевого графа, зокрема, таблиця результатів роботи алгоритмів обчислення мінімального і максимального часу початку виконання та завершення події та беклог із користувацькими історіями (критичним шляхом та резервами). На рисунку 3.11 представлений приклад цього вікна.

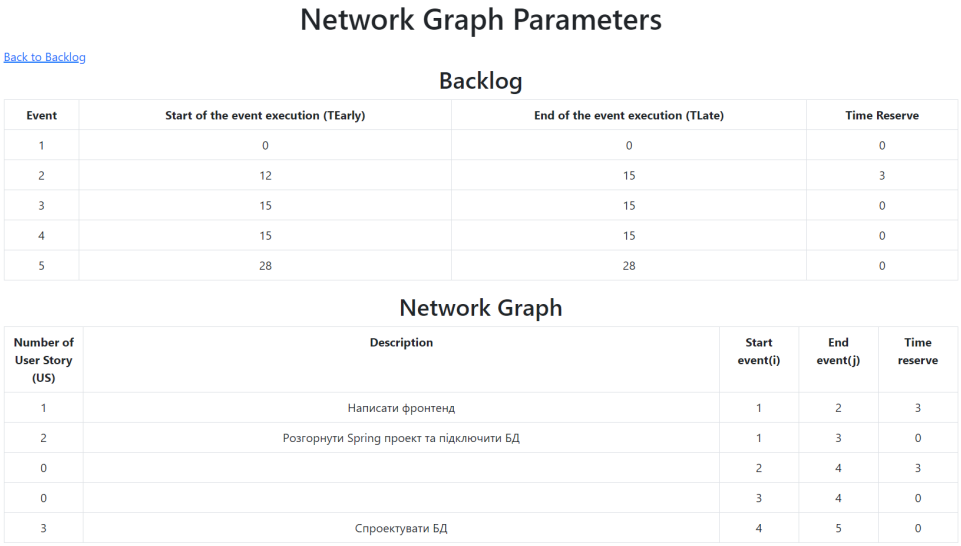



Рисунок 3.11 – Приклад вікна, з відображеними результатами роботи розроблених алгоритмів обходу мережевого графа

З наявного вікна можна спостерігати, що подія два має резерв часу її початку у два сторі пойнта, а користувацька історія під номером один має часовий резерв у три сторі пойнта (фіктивну роботу нуль не беремо до уваги, оскільки вона не несе ніякої цінності для проєкту, а лише була необхідна для побудови мережевого графа). Відтак, критично важливими для проекту є користувацькі історії, у яких часовий резерв рівний нулю, тобто у прикладі це історії під номером два та три.

На рисунку 3.12 зображена графічна інтерпретація мережевого графа етапу планування спринта.

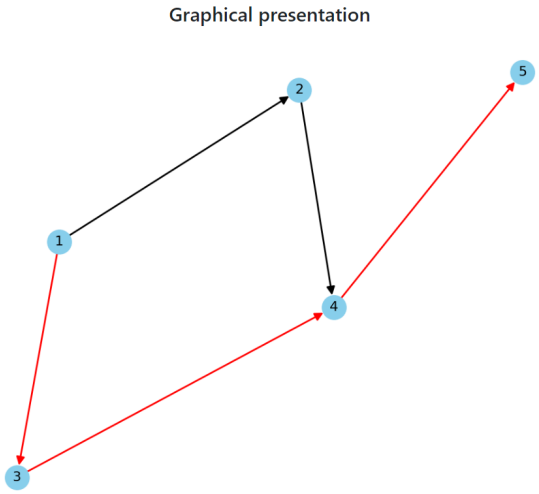
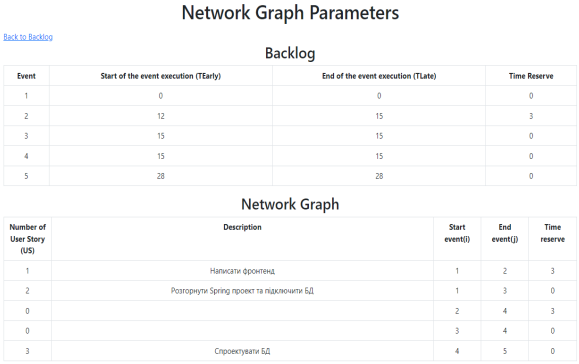


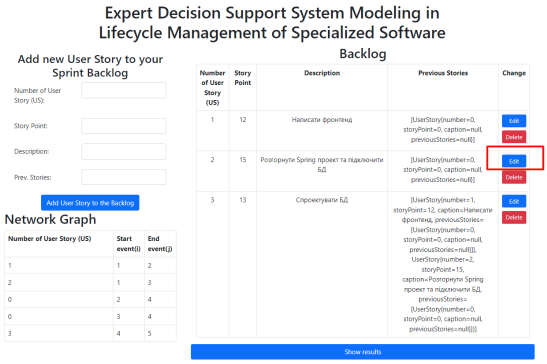

Рисунок 3.12 – Графічне представлення мережевого графа планування (критичний шлях)

Код для реалізації даного функціоналу був написаний мовою програмування Python, зокрема, із використанням таких бібіліотек, як matplotlib, networkx, io та numpy. Спершу був створений невеликий Flask додаток, який обробляв запити, виконував скрипт на Python і повертав зображення графа, а потім за допомогою Python API, зокрема використовуючи RestTemplate для відправки даних до Flask, було отримано зображення графа на вебсторінці.

Враховуючи отримані результати, а також хід виконання спринта, команда розробників може динамічно змінювати оцінку (що у свою чергу змінює час виконання) користувацької історії. Для цього функціоналом програми передбачено кнопку «Back to Backlog», натиснувши на яку користувач повертається на головну сторінку і може редагувати беклог спринта. Приклад зображено на рисунку 5.13.

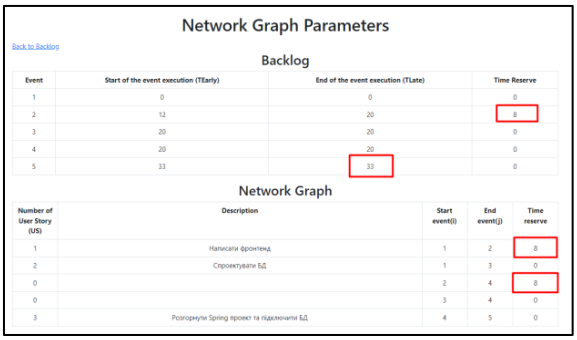



а) – 1 етап (вибір кнопки “Back to Backlog”)




б) 2 етап (вибір кнопки “Edit”)


в) – 3 етап (редагування даних)




г) – 4 етап (перегляд отриманих результатів)

Рисунки 3.13 – Процес перепланування проєкту

Ця інформаційна система дозволяє здійснювати перепланування проекту стільки разів, скільки потрібно команді без втрати часових та матеріальних ресурсів.

На основі розглянутої архітектури та принципів побудови інформаційної системи підтримки прийняття рішень можна виділити такі її ключові переваги:

- Автоматизація розрахунків. Система інтегрує розроблені алгоритми обходу мережевого графа, що дозволяє автоматично розраховувати ранні/пізні терміни виконання подій та визначати часові резерви.

- Динамічне перепланування. Реалізація у форматі web-застосунку забезпечує можливість швидко вносити зміни до беклогу, динамічно переоцінювати тривалість робіт (у Story Point) та оперативно здійснювати перепланування проєкту стільки разів, скільки потрібно, без втрати ресурсів.
- Критичний контроль. ІСППР дозволяє швидко визначати критичний шлях (завдання з нульовим резервом) та обчислювати повний, вільний та незалежний резерви робіт, забезпечуючи ефективний контроль за термінами виконання спринта та запобігаючи збільшенню загального часу реалізації проєкту.

## 3.3. Апробація інформаційної технології підтримки прийняття рішень

На основі розроблених методів та моделей створено велику кількість безпеко-орієнтованих сервісів. Розглянемо практичне впровадження деяких із них у розроблену інформаційну систему підтримки прийняття рішень в управлінні життєвим циклом спеціалізованого програмного забезпечення.

### 3.3.1. Інформаційно-аналітична система «Інтерактивний інспектор»

Інформаційна система розроблена на замовлення Державної служби України із надзвичайних ситуацій. Працює вона у форматі чат-боту на платформі Telegram. Призначена для оцінки протипожежного стану об’єкта залежно від параметрів його функціонування (визначення кількості вогнегасників, необхідність обладнання системами протипожежного захисту, визначення ступеня ризику від провадження господарської діяльності тощо). Система орієнтована на автоматизацію окремих процесів щодо перевірки протипожежного стану об’єкта і використовується посадовими особами ДСНС України та суб’єктами господарювання. На рисунку 3.14 зображений приклад роботи такої аналітичної системи.

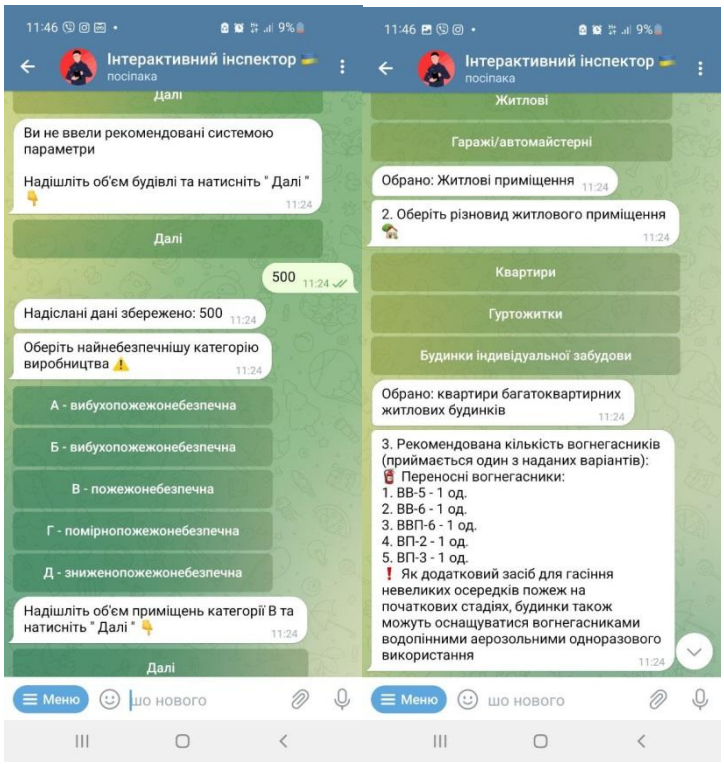
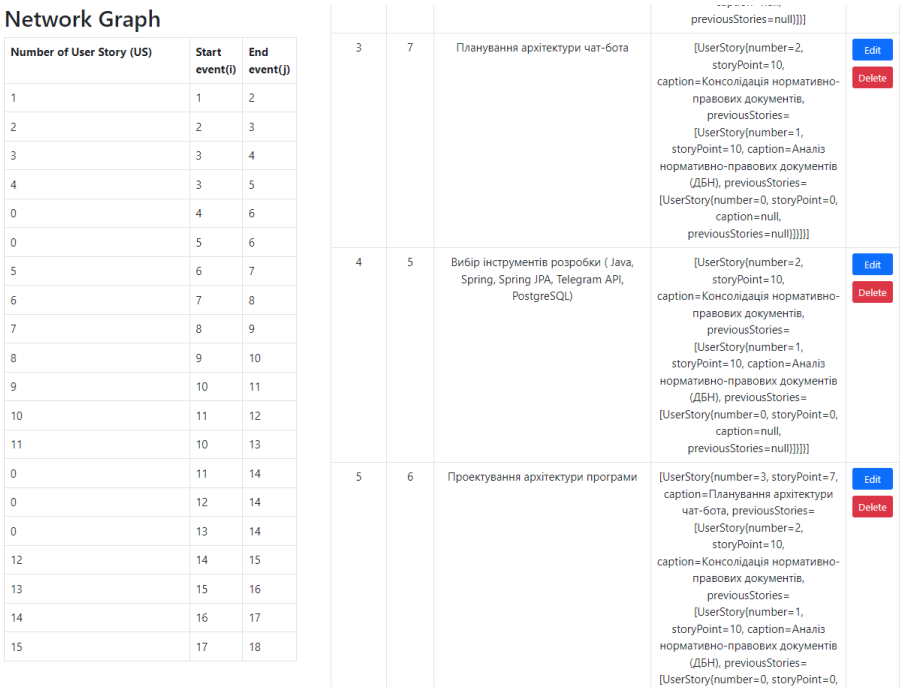



Рисунок 3.14 – Приклад роботи інформаційно-аналітичної системи «Інтерактивний інспектор»

Посилання на систему: **@Interactive_Inspector_Bot**

Вхідні дані та завдання для розробки системи були надані від Державної служби України з надзвичайних ситуацій у період воєнного стану, що підтверджує роботу в динамічних умовах та обмеженість часових ресурсів. На рисунку 3.15 представлений мережевий граф планування одного спринта для розробки даного продукту, котрий занесено до розробленої ІСППР в УЖЦ СПЗ, а на рис 3.16 – приклад його роботи.

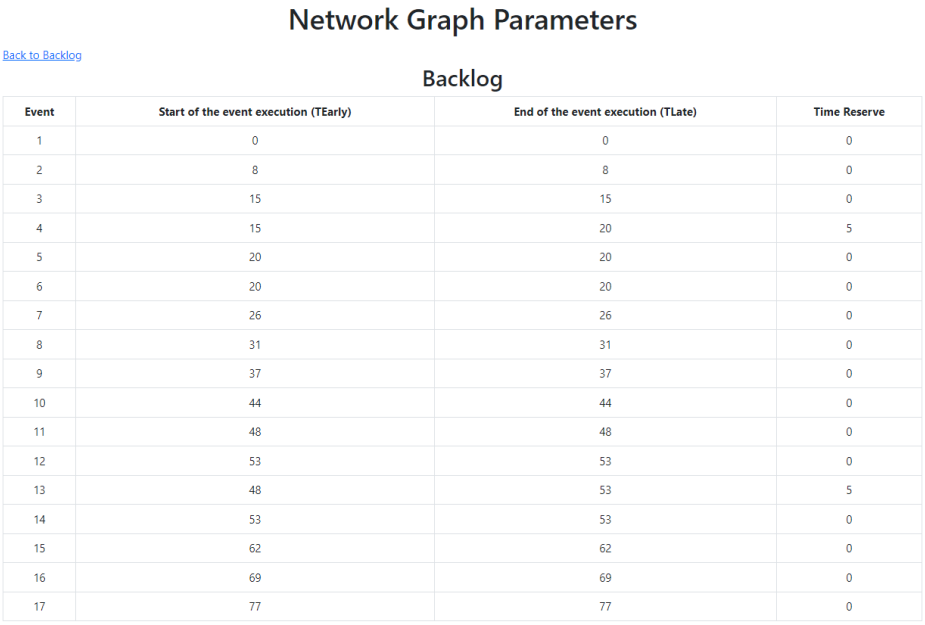


Network Graph

| Number of User Story (US) | Start event(i) | End event(j) |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 3 | 4 |
| 4 | 3 | 5 |
| 0 | 4 | 6 |
| 0 | 5 | 6 |
| 5 | 6 | 7 |
| 6 | 7 | 8 |
| 7 | 8 | 9 |
| 8 | 9 | 10 |
| 9 | 10 | 11 |
| 10 | 11 | 12 |
| 11 | 10 | 13 |
| 0 | 11 | 14 |
| 0 | 12 | 14 |
| 0 | 13 | 14 |
| 12 | 14 | 15 |
| 13 | 15 | 16 |
| 14 | 16 | 17 |
| 15 | 17 | 18 |



Рисунок 3.15 – Аналітичне представлення мережевого графа планування спринта інформаційно-аналітичної системи «Інтерактивний інспектор»



Network Graph Parameters

Back to Backlog

Backlog

| Event | Start of the event execution (TEarly) | End of the event execution (TLate) | Time Reserve |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 8 | 8 | 0 |
| 3 | 15 | 15 | 0 |
| 4 | 15 | 20 | 5 |
| 5 | 20 | 20 | 0 |
| 6 | 20 | 20 | 0 |
| 7 | 26 | 26 | 0 |
| 8 | 31 | 31 | 0 |
| 9 | 37 | 37 | 0 |
| 10 | 44 | 44 | 0 |
| 11 | 48 | 48 | 0 |
| 12 | 53 | 53 | 0 |
| 13 | 48 | 53 | 5 |
| 14 | 53 | 53 | 0 |
| 15 | 62 | 62 | 0 |
| 16 | 69 | 69 | 0 |
| 17 | 77 | 77 | 0 |

Рисунок 3.16 – Час виконання подій та резерви подій ПП

На основі одержаних даних обчислено критично важливі завдання процесу розробки ПП, а також часові резерви для деяких завдань (рисунок 3.17), на рисунку 3.18 графічно продемонстровані результати обчислення критично важливих показників розробки програмного продукту у вигляді графа.

Network Graph

| Number of User Story (US) | Description | Start event(i) | End event(j) | Time reserve |
|---|---|---|---|---|
| 1 | Аналіз нормативно-правових документів (ДБН) | 1 | 2 | 0 |
| 2 | Консолідація нормативно-правових документів | 2 | 3 | 0 |
| 3 | Планування архітектури чат-бота | 2 | 4 | 5 |
| 4 | Вибір інструментів розробки ( Java, Spring, Spring JPA, Telegram API, PostgreSQL) | 3 | 5 | 0 |
| 0 | | 4 | 6 | 5 |
| 0 | | 5 | 6 | 0 |
| 5 | Проектування архітектури програми | 6 | 7 | 0 |
| 6 | Проектування БД | 7 | 8 | 0 |
| 7 | Створення схеми, таблиць БД | 8 | 9 | 0 |
| 8 | Створення проекту, основних класів, підключення Spring | 9 | 10 | 0 |
| 9 | Налаштуваннях з'єднання з БД Postgres за допомогою Spring JPA | 10 | 11 | 0 |
| 10 | Реалізація методів для виконання CRUD операцій з БД | 11 | 12 | 0 |
| 11 | Створення інтерфейсу для взаємодії з користувачем через Telegram API | 10 | 13 | 5 |
| 0 | | 11 | 14 | 5 |
| 0 | | 12 | 14 | 0 |
| 0 | | 13 | 14 | 5 |
| 12 | Реалізація бізнес-логіки для автоматизації процесів оцінки протипожежного стану об'єктів | 14 | 15 | 0 |
| 13 | Виявлення помилок та оптимізація роботи чат бота | 15 | 16 | 0 |
| 14 | Підтримка системи | 16 | 17 | 0 |

Рисунок 3.17 – Беклог спринта та часові резерви

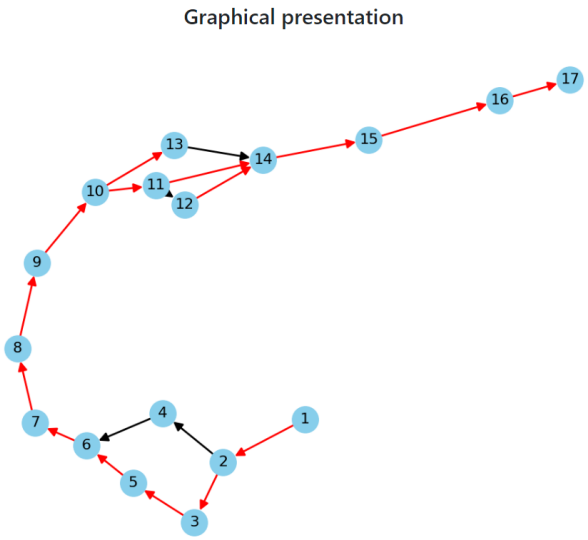



Рисунок 3.18 – Графічно представлені результати обчислення критично важливих показників розробки програмного продукту

Як висновок, команда розробників отримала готовий план, а також могла оцінити, чи зможе виконати його за один спринт, чи потрібно вносити зміни. Варто зазначити, що всі команди розробників, які працювали на даних проєктах є унікальними і, відповідно, кількість сторі пойнтів, які вони можуть взяти на один спринт, для них будуть різними.

### 3.3.2. Вебсервіс обліку пунктів вакцинації та тестування на COVID-19

Ця програмна система була розроблена у 2021 році студентами спеціальності «Комп'ютерні науки». Цей R&D проєкт спрямований на систематизацію і облік пунктів вакцинації та тестування на COVID-19.

Вебзастосунок PCR-Surfer спеціалізується виключно на медичних послугах, які є актуальними для боротьби із COVID-19. Це робить його універсальним та вкрай актуальним в часи боротьби із пандемією.

PCR-Surfer має зручний інтерфейс користувача (рисунок 3.19) та велику кількість можливостей.

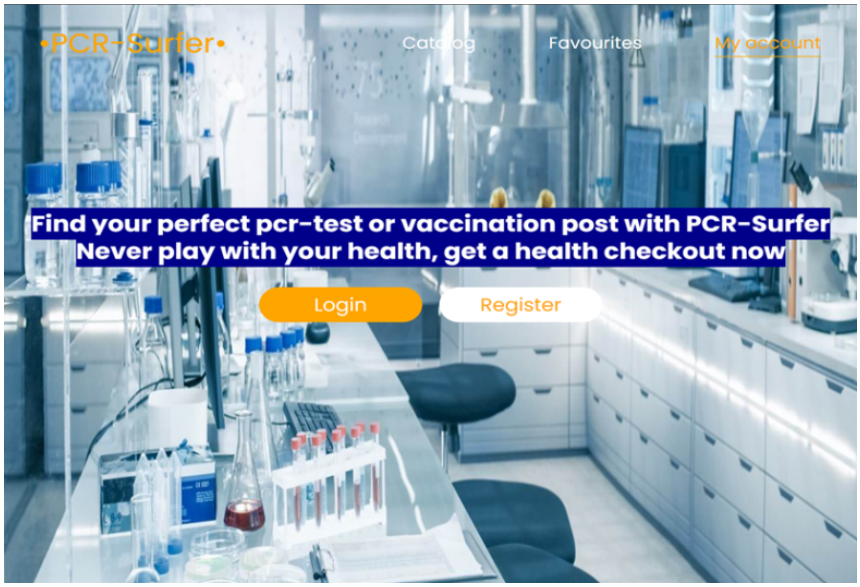
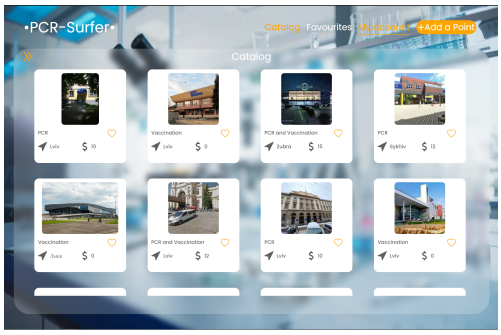


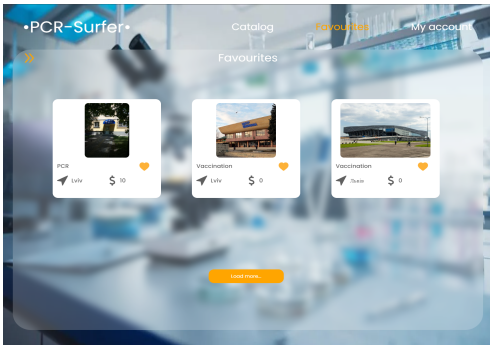

Рисунок 3.19 – Інтерфейс програмної системи PCR-Surfer

Зокрема, серед доступних можливостей користувача є:

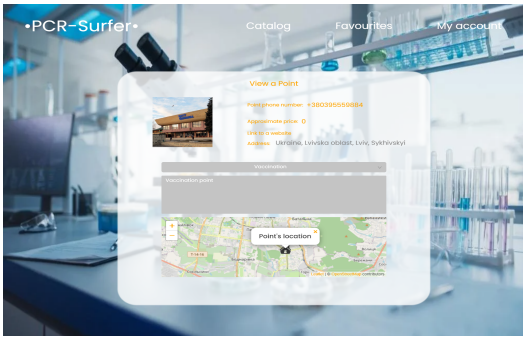
- перегляд великого каталогу пунктів вакцинації та ПЛР-тестування, який постійно оновлюється (рисунок 3.20 а);
- додавання пунктів до уподобаних, з подальшим переглядом (рисунок 3.20 б);
- перегляд точної інформації про пункт: адреса, розташування, номер телефону, опис, посилання на вебсторінку пункту, ціна тощо (рисунок 3.20 в);
- власний профіль користувача (рисунок 3.20 г).



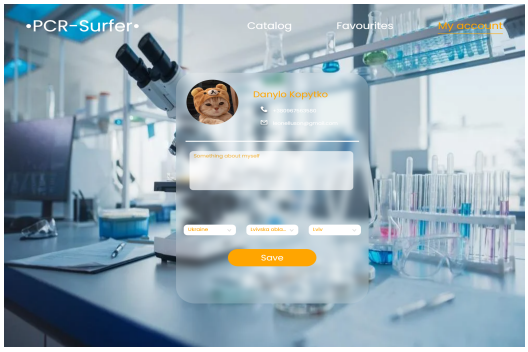

а) – перегляд каталогу




б) додавання до улюблених




в) – детальна інформація про пункт




г) – профіль користувача

Рисунок 3.20 – Функціонал програмної системи PCR-Surfer

Переваги для компаній, які надають послуги у цій сфері:

- додаткова реклама послуги;
- зручний інтерфейс;
- пряме посилання на сайт, що в свою чергу збільшує конверсію.

Особливо актуальним даний вебзастосунок був на початку війни у лютому-березні 2022 року. Враховуючи велику кількість вимушено переміщених осіб ризик захворіти на COVID-19 був значно більший. Ця автоматизована система забезпечувала легкий та зручний пошук потрібного пункту, відслідковування його на карті та, за потреби, порівняння ціни на послугу у різних локаціях району перебування.

Наразі підтримка цього застосунку не відбувається, проте переглянути інтерфейс вебсайту можна за посиланням https://master.d2wa196ubyd6uf.amplifyapp.com/.

На рисунку 3.21 зображений мережевий граф беклогу завдань, занесених у розроблену ІСППР в УЖЦ СПЗ, а на рисунку 3.22 – час виконання подій та резерви подій ПП.

## Network Graph

| Number of User Story (US) | Start event(i) | End event(j) |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 3 |
| 3 | 1 | 4 |
| 4 | 2 | 5 |
| 5 | 2 | 6 |
| 6 | 2 | 7 |
| 7 | 2 | 8 |
| 0 | 3 | 9 |
| 0 | 4 | 9 |
| 8 | 9 | 10 |
| 0 | 8 | 11 |
| 0 | 10 | 11 |
| 9 | 11 | 12 |
| 0 | 5 | 13 |
| 0 | 6 | 13 |
| 0 | 7 | 13 |
| 0 | 12 | 13 |
| 10 | 13 | 14 |
| 11 | 14 | 15 |
| 12 | 15 | 16 |

| | | | | |
|---|---|---|---|---|
| | | | previousStories=null}] | Delete |
| 4 | 7 | FrontEnd розробка головної сторінки | [UserStory{number=1, storyPoint=10, caption=Розробка UI/UX дизайну веб-сайта, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}] | Edit Delete |
| 5 | 6 | FrontEnd розробка вікна реєстрації | [UserStory{number=1, storyPoint=10, caption=Розробка UI/UX дизайну веб-сайта, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}] | Edit Delete |
| 6 | 7 | FrontEnd розробка вікна з каталогом пунктів | [UserStory{number=1, storyPoint=10, caption=Розробка UI/UX дизайну веб-сайта, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}] | Edit Delete |
| 7 | 7 | FrontEnd розробка вікна з інформацією про пункт | [UserStory{number=1, storyPoint=10, caption=Розробка UI/UX дизайну веб-сайта, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}] | Edit Delete |
| 8 | 9 | Створення моделей адміністратора, користувача та пункта | [UserStory{number=2, storyPoint=8, caption=Розгортання проєкта, налаштування сервера, previousStories= [UserStory{number=0, | Edit Delete |

Рисунок 3.21 – Аналітичне представлення мережевого графа

## Network Graph Parameters

Back to Backlog

### Backlog

| Event | Start of the event execution (TEarly) | End of the event execution (TLate) | Time Reserve |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 10 | 19 | 9 |
| 3 | 8 | 9 | 1 |
| 4 | 9 | 9 | 0 |
| 5 | 17 | 26 | 9 |
| 6 | 16 | 26 | 10 |
| 7 | 17 | 26 | 9 |
| 8 | 17 | 18 | 1 |
| 9 | 9 | 9 | 0 |
| 10 | 18 | 18 | 0 |
| 11 | 18 | 18 | 0 |
| 12 | 26 | 26 | 0 |
| 13 | 26 | 26 | 0 |
| 14 | 41 | 41 | 0 |
| 15 | 55 | 55 | 0 |
| 16 | 65 | 65 | 0 |

Рисунок 3.22 – Час виконання подій та резерви подій ПП

На основі цих даних обчислено критично важливі завдання процесу розробки даного ПП, а також часові резерви для деяких завдань (рисунок 3.23) та їх графічне представлення у вигляді графа (рисунок 3.24).

Network Graph

| Number of User Story (US) | Description | Start event(i) | End event(j) | Time reserve |
|---|---|---|---|---|
| 1 | Розробка UI/UX дизайну веб-сайта | 1 | 2 | 9 |
| 2 | Розгортання проєкта, налаштування сервера | 1 | 3 | 1 |
| 3 | Проєктування бази даних | 1 | 4 | 0 |
| 4 | FrontEnd розробка головної сторінки | 2 | 5 | 9 |
| 5 | FrontEnd розробка вікна реєстрації | 2 | 6 | 10 |
| 6 | FrontEnd розробка вікна з каталогом пунктів | 2 | 7 | 9 |
| 7 | FrontEnd розробка вікна з інформацією про пункт | 2 | 8 | 1 |
| 0 | | 3 | 9 | 1 |
| 0 | | 4 | 9 | 0 |
| 8 | Створення моделей адміністратора, користувача та пункта | 9 | 10 | 0 |
| 0 | | 8 | 11 | 1 |
| 0 | | 10 | 11 | 0 |
| 9 | Підключення бібліотеки Leaflet для відображення карт (та маршутів) на сторінці "Інформація про пункт" | 11 | 12 | 0 |
| 0 | | 5 | 13 | 9 |
| 0 | | 6 | 13 | 10 |
| 0 | | 7 | 13 | 9 |
| 0 | | 12 | 13 | 0 |
| 10 | Реалізація CRUD методів для ролі Адміністратор при роботі із пунктами | 13 | 14 | 0 |
| 11 | Реалізація функціоналу для користувача: перегляд пункту, додавання до улюбленого, перегляд улюблених пунктів | 14 | 15 | 0 |
| 12 | Тестування та підтримка системи | 15 | 16 | 0 |

Рисунок 3.23. – Беклог спринта та часові резерви

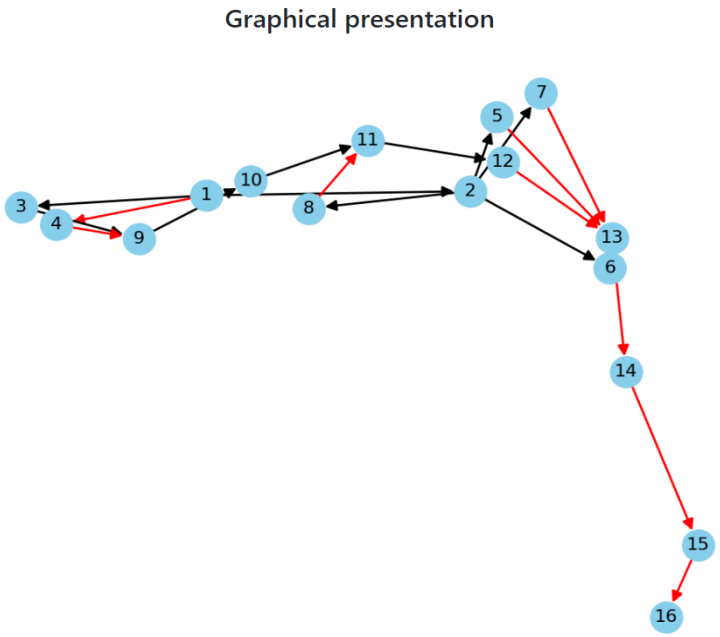



Рисунок 3.24 – Графічне представлення та критично важливі завдання проєкту

### 3.3.3. Інформаційна система «Я-Доброволець»

Інформаційна система «Я-Доброволець» спрямована на створення інноваційної платформи для організації та координування дій соціально активних громадян, які готові долучитись до ліквідації наслідків надзвичайних подій (на допомогу основним підрозділам).

Мета проєкту: створення безпекового середовища громади шляхом об’єднання зусиль добровольців, а також небайдужих громадян, які мають відповідний досвід та бажання допомогти.

Ця програмна система, наразі, на стадії розробки, тому ознайомитись із функціоналом практично, наразі немає можливості. На рисунку 3.25 можна переглянути інтерфейс системи, який наразі розробляється та на рисунку 3.26 представлене аналітичне зображення мережевого графа для окремого спринта проєкту. На рисунку 3.27 зображений час виконання подій та їх резерви, рисунок 3.28 містить беклог спринта та часові резерви, а на рисунку 3.29 графічно представлені критично важливі завдання проєкту.

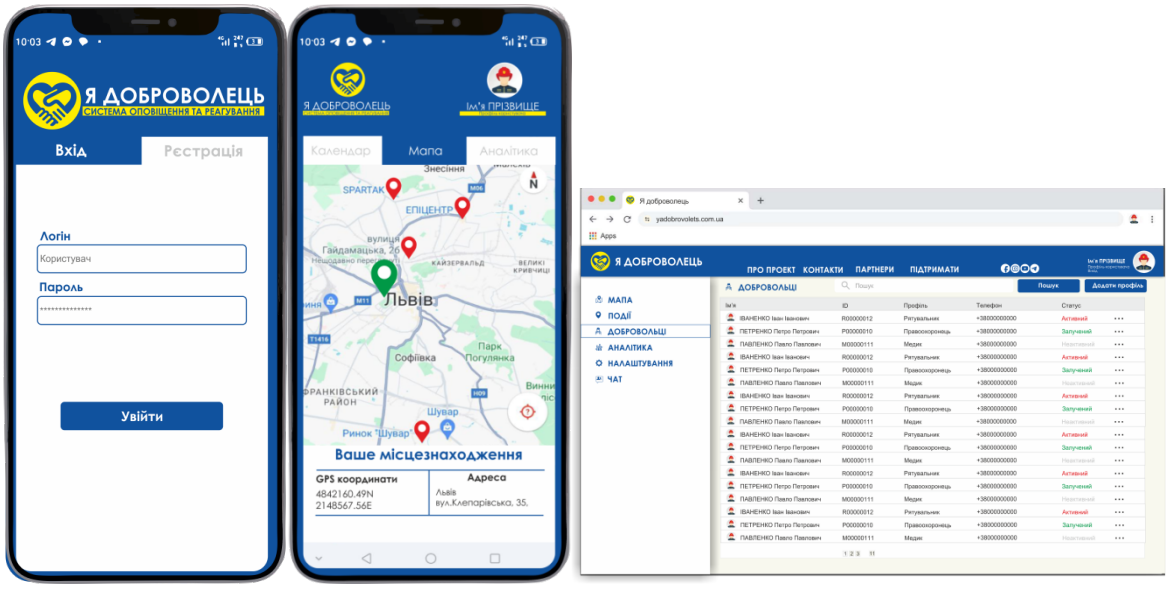



Рисунок 3.25 – Інтерфейс системи «Я-Доброволець»

## Network Graph

| Number of User Story (US) | Start event(i) | End event(j) |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 3 |
| 3 | 1 | 4 |
| 4 | 1 | 5 |
| 0 | 2 | 6 |
| 0 | 3 | 6 |
| 0 | 4 | 6 |
| 0 | 5 | 6 |
| 5 | 6 | 7 |
| 6 | 7 | 8 |
| 7 | 8 | 9 |
| 8 | 8 | 10 |
| 9 | 8 | 11 |
| 0 | 9 | 12 |
| 0 | 10 | 12 |
| 0 | 11 | 12 |
| 10 | 12 | 13 |
| 11 | 13 | 14 |
| 12 | 14 | 15 |

| | | | | |
|---|---|---|---|---|
| | | проекту | previousStories=null}] | Delete |
| 4 | 9 | Розробити структуру БД для volunteer-core | [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}] | Edit Delete |
| 5 | 10 | Створити CRUD API для Volunteer, Event | [UserStory{number=1, storyPoint=8, caption=Створити і налаштувати репозиторії, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}, UserStory{number=2, storyPoint=8, caption=Створити скелетон проекту volunteer-core, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}, UserStory{number=3, storyPoint=8, caption=Створити документацію по Архітектурі проекту, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}, UserStory{number=4, storyPoint=9, caption=Розробити структуру БД для volunteer-core, previousStories= [UserStory{number=0, storyPoint=0, caption=null, previousStories=null}]}] | Edit Delete |
| 6 | 9 | Додати OpenAPI volunteer-core | [UserStory{number=5, storyPoint=10, caption=Створити CRUD API для Volunteer, Event, | Edit Delete |

Рисунок 3.26 – Аналітичне представлення мережевого графа

## Network Graph Parameters

Back to Backlog

### Backlog

| Event | Start of the event execution (TEarly) | End of the event execution (TLate) | Time Reserve |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 8 | 9 | 1 |
| 3 | 8 | 9 | 1 |
| 4 | 8 | 9 | 1 |
| 5 | 9 | 9 | 0 |
| 6 | 9 | 9 | 0 |
| 7 | 19 | 19 | 0 |
| 8 | 28 | 28 | 0 |
| 9 | 36 | 36 | 0 |
| 10 | 35 | 36 | 1 |
| 11 | 35 | 36 | 1 |
| 12 | 36 | 36 | 0 |
| 13 | 44 | 44 | 0 |
| 14 | 54 | 54 | 0 |
| 15 | 61 | 61 | 0 |

Рисунок 3.27 – Час виконання подій та резерви подій ПП

## Network Graph

| Number of User Story (US) | Description | Start event(i) | End event(j) | Time reserve |
|---|---|---|---|---|
| 1 | Створити і налаштувати репозиторії | 1 | 2 | 1 |
| 2 | Створити скелетон проекту volunteer-core | 1 | 3 | 1 |
| 3 | Створити документацію по Архітектурі проекту | 1 | 4 | 1 |
| 4 | Розробити структуру БД для volunteer-core | 1 | 5 | 0 |
| 0 | | 2 | 6 | 1 |
| 0 | | 3 | 6 | 1 |
| 0 | | 4 | 6 | 1 |
| 0 | | 5 | 6 | 0 |
| 5 | Створити CRUD API для Volunteer, Event | 6 | 7 | 0 |
| 6 | Додати OpenAPI volunteer-core | 7 | 8 | 0 |
| 7 | Реалізація процесу пошуку волонтерів для направлення на подію | 8 | 9 | 0 |
| 8 | Implement VolunteerRegistryService | 8 | 10 | 1 |
| 9 | Implement VolunteerRegistryClient | 8 | 11 | 1 |
| 0 | | 9 | 12 | 0 |
| 0 | | 10 | 12 | 1 |
| 0 | | 11 | 12 | 1 |
| 10 | Implement VolunteerRegistryClient findById | 12 | 13 | 0 |
| 11 | Fix FirebaseInitializer volunteer-messaging-service | 13 | 14 | 0 |
| 12 | Підготувати контроллер для моб апки який буде повертати всю необхфдну інформацію про подію для волонтера | 14 | 15 | 0 |

Рисунок 3.28 – Беклог спринта та часові резерви

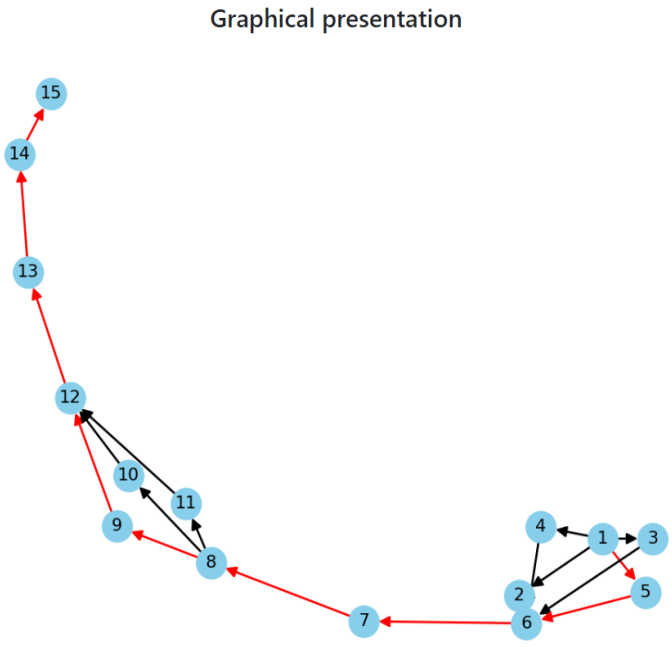



Рисунок 3.29 – Графічне представлення критично важливих завдань проєкту

### 3.4. Висновки

Результати розділу наводять на такі висновки:

1. Шляхом алгоритмічного відтворення розроблених імітаційних моделей побудовано чітку процедуру розрахунку раннього та пізнього термінів виконання подій, що дає змогу провести автоматизацію цих процесів для вирішення задач короткострокового планування проєктів з розроблення спеціалізованого програмного забезпечення. А також розроблено алгоритми обходу мережевого гафа та визначення часових резервів.

2. Розроблено інформаційну систему підтримки прийняття рішень (ІСППР), яка інтегрує вказані методи, моделі та алгоритми. Система забезпечує автоматизацію процесів короткострокового планування розробки безпеко-орієнтованих сервісів та дозволяє швидко й якісно переоцінювати тривалість робіт при внесенні будь-яких змін до змісту, обсягу, часу виконання спринтів або складу команди.

3. Здійснено успішну апробацію ІСППР, підтвердивши її ефективність в управлінні життєвим циклом реальних безпеко-орієнтованих сервісів, зокрема: «Інтерактивний інспектор», вебсервіс обліку пунктів вакцинації та тестування на COVID-19, та інформаційна система «Я-Доброволець».

# ВИСНОВКИ

У роботі на підставі проведених досліджень розв’язано важливу науково-прикладну задачу підвищення ефективності створення спеціалізованого програмного забезпечення шляхом розроблення нових моделей та засобів адаптивного планування життєвого циклу в динамічних умовах.

Основні результати є такими:

1. У результаті проведеного аналізу предметної області та літературних джерел обгрунтовано вплив гнучких методів на інновації спеціалізованого програмного забезпечення, систематизовано основні концепції методів гнучкої методології управління життєвим циклом програмних систем, розширено існуючі емпіричні дані про можливість та переваги використання гнучких методів у безпековій галузі, що зумовило передумови необхідності розроблення нових методів та моделей управлінням життєвим циклом розроблення спеціалізованого програмного забезпечення.

2. Шляхом моделювання процесу розробки програмних систем із використанням понятійного апарату теорії множин отримано математичний опис їх життєвого циклу, який дає змогу охарактеризувати обсяги робіт на окремих етапах розроблення програмного продукту, встановити взаємозв’язки і взаємозалежності між ними та сформувати фундаментальні принципи автоматизації процедури підтримки прийняття управлінських рішень на різних етапах розроблення програмних систем, зокрема, безпеко-орієнтованого спрямування в динамічному оточенні.

3. Розроблено концептуальну модель процесу управління життєвим циклом спеціалізованого програмного забезпечення (безпеко-орієнтованих сервісів), котра адаптована під специфіку роботи Державної служби України із надзвичайних ситуацій та корелює із принципами гнучкої методології управління життєвим циклом програмного забезпечення.

4. Шляхом імітаційного моделювання з використанням понятійного апарату мереж Петрі отримано уніфіковану модель процесів обходу мережевих графів

планування, яка полягає у циклічності процедури встановлення зв’язку між попередньою і поточною подіями та може використовуватися незалежно від складності та конфігурації мережевих графів.

5. За результатами оптимізації математичних методів мережевого планування під процеси управління життєвим циклом спеціалізованого програмного забезпечення, а також розроблених імітаційних моделей процесів обходу мережевого графа, розроблено алгоритми побудови та обходу графа мережевої моделі для визначення її основних параметрів, що надають можливість підвищити ефективність планування окремих етапів розробки програмного забезпечення та здійснювати їх коригування в режимі реального часу.

6. Розроблено інформаційну технологію підтримки прийняття рішень в процесі управління життєвим циклом розроблення спеціалізованого програмного забезпечення шляхом впровадження у неї розроблених моделей та алгоритмів для вирішення задачі короткострокового планування розробки безпеко-орієнтованих сервісів. Дана ІС здійснює автоматизацію процесів короткострокового планування, а також швидко і якісно переоцінює тривалість робіт за умови введення будь яких змін до змісту, обсягу, часу виконання окремих спринтів проєкту або складу команди розробників.

7. Здійснено програмну реалізацію та апробацію розробленої інформаційної технології підтримки прийняття рішень під час розробки безпеко-орієнтованих сервісів для потреб цивільного захисту.

# ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Agile-маніфест розробки програмного забезпечення [Електронний ресурс] URL: https://agilemanifesto.org/iso/uk/manifesto.html.

2. Алексієв В. О. Визначення методів та засобів для уніфікації процесу розробки веб-ресурсів малого підприємства. *Матеріали VI міжнародної науково-технічної Інтернет-конференції "Автомобіль і електроніка. сучасні технології*", 19-20 листопада 2018 р. Харків. 2018. С. 48-50.

3. Алексієв В. О., Труш О. М. Аналіз методів та оптимізація процесу розробки програмних продуктів. *Моделювання та інформаційні технології в науці, техніці та освіті: зб. наук. праць міжнар. наук.-практ. Internet-конф*. 21-22 лист. 2018 р. Харків, 2018. С. 45-51.

4. Асєєва А. В., Кулаковська І. В. Аналіз проблем вибору технології для розробки програмного забезпечення. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. Луцьк, 2019. №37. С. 10 – 18.

5. Близнюкова І. О., Семко С. Г., Кійко С. Г. Огляд сучасних методологій управління командами ІТ-проєктів. *Управління розвитком складних систем*. Київ, 2020. № 43. С. 60 – 66. https://doi.org/10.32347/2412-9933.2020.43.60-66

6. Бобрицька Г.С. Прикладне застосування теорії графів у різних сферах життя суспільства та окремої особистості. *Фізико-математична освіта*. 2017. В. 3(13). С. 26-30.

7. Богач І., Опольський Я. Застосування програмної бібліотеки машинного навчання TensorFlow для реалізації розпізнавання вмісту зображень на мобільних платформах : дис. – ВНТУ, 2018.

8. Бушуєв С.Д., Бушуєва В. Б., Бойко О. О. Agile- трансформація підходів в управлінні будівельними проектами, фазах ініціалізації та проєктування. *Управління розвитком складних систем*. Київ, 2020. №41. С. 14 – 20. https://doi.org/10.32347/2412-9933.2020.41.15-20

9. Вавіленкова А. І. Аналіз гнучких методологій розробки програмного забезпечення для реалізації у командних проєктах. *Вісник Національного*

*технічного університету «ХПІ»*. Харків, 2021. № 1(7). С. 39 – 46. https://doi.org/10.20998/2413-4295.2021.01.06

10. Ванін В.В., Залевська О.В., Яблонський П.М. Застосування теорії графів для удосконалення та візуалізації алгоритму пошуку найкоротшого шляху в математичній моделі відео ігри. *Прикладна геометрія та інженерна графіка*, 2020 №97, с. 23-28

11. Великодний С. С., Бурлаченко Ж. В., Зайцева-Великодна С. С. Розробка архітектури програмного засобу для управління мережевим плануванням реінжинірингу програмного проекту. *Сучасний стан наукових досліджень та технологій в промисловості*. 2019. № 2 (8). С. 25–35. DOI: https://doi.org/10.30837/2522-9818.2019.8.025

12. Возняк А. Науково-дослідницька діяльність студентів як важливий напрям роботи з обдарованою молоддю. *Вища школа. Гуманізація навчально-виховного процесу*. 2011. №7. С. 77–82

13. Герасимчук О. Алгоритми розпізнавання символів для систем штучного інтелекту. *Матеріали X студентської науково-технічної конференції «Природничі та гуманітарні науки. Актуальні питання»*. 2007. С. 117-117.

14. Данченко О. Б. Методи та засоби аналізу проектних ризиків. *Вісник Черкаського державного технологічного університету* . Черкаси. 2004, № 1. С. 87-92.

15. Димова Г.О., Ларченко О. В. Розробка комп'ютерної програми розв'язання задач мережевої оптимізації. *Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво"*. Луцьк, 2020. Випуск № 41 С. 143–151

16. Єчина Ю. С. Науково-дослідницька діяльність студентів як підгрунтя науково-технічного розвитку. *Вісник КНУТД*. 2012. №5 С. 341-347.

17. Кім О. О., Козлова В. В. Перспективи застосування методології Agile менеджменту в управлінні ІТ-проектами. *Соціальна економіка*. Харків. 2019. № 58. С. 95–99. https://doi.org/10.26565/2524-2547-2019-58-12

18. Клочко, Н. Б., Чеховський, С. А., & Слабінога, М. О. Дослідження фізичної моделі процесу вимірювання об’єму газу турбінними лічильниками методом Монте-Карло. *Математичне моделювання*. №2, 2016. С. 77-79.

19. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розробки програмного забезпечення. *Системи управління, навігації та зв’язку*. Полтава, 2018. Т. 3(49). С. 116-125. https://doi.org/10.26906/SUNZ.2018.3.116

20. Коваль М. С., Литвин А. В. Завдання та властивості інформаційно-освітнього середовища закладу вищої освіти ДСНС України. *Модернізація змісту професійної освіти в умовах євроінтеграції України : матеріали Всеукраїнської науково-практичної конференції*. Київ, 2021. Ч.1. С. 39-43.

21. Ковальчук О. І., Зачко О. Б. Моделі життєвого циклу розвитку проєктних команд в системі цивільного захисту. *Вісник Львівського державного університету безпеки життєдіяльності*. Львів, 2022. №25. С. 71-78. https://doi.org/10.32447/20784643.25.2022.08

22. Козир І. С. Фактори впровадження Agile-менеджменту в практику управління. *I International Scientific and Practical Conference «Problemas y perspectivas de la aplicación de la investigación científica innovadora»*. Кембридж. 2021. Т. 1. С. 78-79. https://doi.org/10.36074/logos-19.03.2021.v1.26

23. Козяр М. М., Козловський Ю. М., Стечкевич О. О. Реалізація можливостей Stem-освіти засобами інтеграції креативних методів навчання. *Наукові записки. Педагогічні науки.* Кропивницький, 2020. № 191. С. 20-23.

24. Колянко О. В., Озимок Г. В., Використання жорсткої "Waterfall" та гнучкої "Agile" моделей управління проектами. *Вісник Львівського торговельно-економічного університету. Економічні науки*. Львів, 2017. Вип. 52. С. 177 – 182.

25. Кордунова Ю. С., Придатко О. В., Смотр О. О. Переваги використання Agile- методології під час розробки програмного забезпечення в умовах сучасного ринку. *Інформаційна безпека та інформаційні технології : зб. наук. праць IV Всеукр. наук.-практ. конф. молодих учених, студентів і курсантів.* м. Львів 27 листопада 2020 р. Львів, 2020. С. 206 – 207

26. Кордунова Ю. С., Смотр О. О. Визначення ефективності використання Agile методології в сучасних організаціях. *Проблеми та перспективи забезпечення цивільного захисту: матеріали міжнародної науково-практичної конференції молодих учених*. Харків: НУЦЗУ, 2021. С. 166.

27. Кордунова Ю. С., Смотр О. О. Сенс Agile-маніфесту для сучасного проєкт-менеджменту. *Проблеми та перспективи розвитку системи безпеки життєдіяльності: зб. наук. праць XVI Міжнар. наук.-практ. конф. молодих вче1них, курсантів та студентів*. – Львів: ЛДУ БЖД, 2021. С. 247-248

28. Кордунова Ю. С., Смотр О. О., Кокотко І. Я., Малець Р. Б. Аналіз традиційного та гнучкого підходів до створення програмного забезпечення в динамічних умовах. *Управління розвитком складних систем*. Київ, 2021. № 47. С. 71 – 77, https://doi.org/10.32347/2412-9933.2021.47.71-77

29. Кордунова Ю.С., Придатко О.В., Смотр О.О. Переваги використання Agile-методології під час розробки програмного забезпечення в умовах сучасного ринку. *Інформаційна безпека та інформаційні технології*, Вип.4, 206-207.

30. Кордунова, Ю., Фелтіновскі, М., Придатко, О., Смотр, О. Математичне моделювання процесу розробки спеціалізованих програмних систем безпеко-орієнтованого спрямування. *Вісник Львівського державного університету безпеки життєдіяльності*. Львів, 2023. № 27, С. 23-31. https://doi.org/https://doi.org/10.32447/20784643.27.2023.03

31. Кузь М. В, Пікуляк М. В. Остафійчук Т. Д. Модель системи управління якістю процесу розробки програмного забезпечення. *Proceedings of the 2019 Scientific Seminar on Innovative Solutions in Software Engineering*. Івано-Франківськ, 2019. С. 19-21, https://doi.org/10.5281/zenodo.4091480

32. Кузьменко А. І., Коциловський М. П. Удосконалення організації руху вантажопотоків у транспортній системі України. Системи та технології, № 2 (54), 2015 с. 81–89

33. Кузьменок та ін. Алгоритми пошуку найкоротших шляхів та їх застосування в комп'ютерних іграх. Комп’ютерні системи та інформаційні технології. 2021. № 2. С. 78-84.

34. Латанська Л. О., Балашов М. О. Удосконалення регресійного рівняння для оцінювання трудомісткості розробки програмного забезпечення, створеного за гнучкою методологією. *Матеріали XI-ої Міжнародної науково-практичної конференції «Free and Open Source Software»*, Харків, 19-21 листопада 2019 р. – Харків: Харківський національний університет будівництва та архітектури, С. 46.

35. Муравецький С. А., Крамський С. О. Планування процесів забезпечення якості у великих та географічно розподілених гібридних IT-проектах. *Вісник НТУ «ХПІ»*. Харків, 2016. №1(1173). С. 106–109. https://doi.org/10.20998/2413-3000.2016.1173.21

36. Павлова О.О., Лопатто І.Ю., Говорущенко Т.О. Метод діяльності та структура інтелектуального агента на основі онтологічного підходу для оцінювання початкових етапів життєвого циклу програмного забезпечення. *Вісник Хмельницького національного університету*, Хмельницький, 2020. №3. С.61-64

37. Пасічник А. М., Кутирєв В. В., Бугерко К. М. Розрахунок максимального потоку вантажів у транспортній мережі шляхом застосування алгоритму Форда–Фалкерсона. *Вісник АМСУ. Серія: "Технічні науки"*, № 2 (52), 2014 с. 18–31

38. Праворська Н., Мартинюк В. Конструювання програмного забезпечення за допомогою синхронного підходу: основні процеси та інструменти для ефективної реалізації Devops. *Вісник Хмельницького національного університету*, Том 1, №5, 2023 С. 182-191

39. Придатко О. В., Кордунова Ю.С, Кокотко І. Я., Головатий Р. Р. Обгрунтування методології управління студентськими R&D проєктами (на прикладі освітньої програми комп'ютерні науки) *"Інформаційно-комунікаційні технології в сучасній освіті: досвід, проблеми, перспективи"* Львів, 2021.

40. Придатко О. В., Придатко В. В., Борзов Ю. О., Дзень В. Є. Інтеграція новаційного методу мобільного навчання в освітні проекти підготовки розробників програмного забезпечення. *Вісник Львівського державного університету безпеки життєдіяльності*, Львів: ЛДУ БЖД, 2018. №18. С.70-80.

41. Приймак В. Гнучкі моделі управління командною роботою інжинірингових проектів. *Вісник Київського національного університету мені Тараса Шевченка. Економіка*. Київ, 2019. №6 (207). С. 21-27. https://doi.org/10.17721/1728-2667.2019/207-6/3

42. Прошкін В. В. Стимулювання студентських наукових пошуків як засіб інтеграції науки й освіти. *Професійна освіта. Педагогіка*. 2010. № 1. С.39–44.

43. Семенов С. Г., Халифе Кассем, Захарченко М. М. Удосконалений спосіб масштабування гнучкої методології розробки програмного забезпечення. . Вісник НТУ «ХПІ». Харків, 2017. Т. 1, № 1. С. 79–84. https://doi.org/10.20998/2522-9052.2017.1.15

44. Сєдих О.Л., Чобану В.В. Оптимізація мережевого графіка комплексу робіт. *Modern engineering and innovative technologies.* № 03-01, 2018 С. 61-67 https://doi.org/10.30890/2567-5273.2018-03-01-009 10.

45. Сидорчук Н.Г. До питання про організацію науково-дослідної роботи студентів педагогічних навчальних закладів. *Зб.наук.пр. Сучасні інформаційні технології та інноваційні методики навчання у підготовці фахівців: методологія, теорія, досвід, проблеми*. Київ-Вінниця, 2002. С. 408-413.

46. Студентський R&D проєкт [Електронний ресурс] URL: https://www.slideshare.net/GlobalLogicUkraine/rd-236853435

47. Теорія планування експериментів: Виконання розрахунково-графічної роботи [Електронний ресурс] : навч. посіб. для студ. спеціальності 131 «Прикладна механіка», спеціалізації «Технологія машинобудування». Київ : КПІ ім. Ігоря Сікорського,2020. 86 с.

48. Токарєв В. В., Славтіч Д. О. Застосування алгоритма Дейкстри при проектуванні «розумних зупинок». *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення: збірник тез доповідей*, 16 лист. 2020р, Тернопіль, 2020. В. 53. Ч.1. С.100-101.

49. Шапошнікова О.П., Кірвас В.В. Застосування методології Agile в практиці проектного навчання при підготовці ІТ спеціалістів. *Системи обробки*

*інформації*. Харків. 2020. № 4(163). С. 94-100. https://doi.org/10.30748/soi.2020.163.10

50. Якубенко І. М. Agile-менеджмент, як дієве управління проектами для цілеспрямованих команд. *Економіка. Менеджмент. Бізнес*. 2017. №4(22). С. 167–172.

51. A guide to the Project Management Body of Knowledge. PMBOK guide SIXTH EDITION – USA: Project Management Institute, 2017.

52. Adhikary, S. Images Within Images? A Multi-image Paradigm with Novel Key-Value Graph Oriented Steganography. *Intelligent Computing & Optimization. ICO 2021. Lecture Notes in Networks and Systems*, 2022. vol 371. Springer, Cham. https://doi.org/10.1007/978-3-030-93247-3_83

53. Anand R. Vijay, Dinakaran M. Improved scrum method through staging priority and cyclomatic complexity to enhance software process and quality. *International Journal of Internet Technology and Secured Transactions*, 8 (2), 2018. p.p. 150-166. https://doi.org/10.1504/IJITST.2018.093342

54. Avasthi A. & Mishra G. A New Framework for the Agile Software Development Method. *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, p.p. 436-438, https://doi.org/10.1109/ICECA.2018.8474737.

55. Barbareschi M., Barone S., Carbone R. et al. Scrum for safety: an agile methodology for safety-critical software systems. *Software Qual J.* 2022. №30, pp. 1067–1088 https://doi.org/10.1007/s11219-022-09593-2

56. Barraood S. O., Mohd H., Baharom F. A Comparison Study of Software Testing Activities in Agile Methods. *Knowledge Management International Conference (KMICe) Virtual Conference*. Malaysia, 2021. pp. 130-137

57. Belkasmi, M.G., Bougroun, Z., Farissi, I.E., Emharraf, M., Belouali, S., Chadli, S., Saber, M.: Global IT project management: An agile planning assistance. *Advances in Smart Technologies Applications and Case Studies*, pp. 575–582. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-53187-4_63

58. Benedicenti L., Messina A. & Sillitti A., IAgile: Mission Critical Military Software Development. *International Conference on High Performance Computing & Simulation (HPCS)*, 2017. p.p. 545-552, https://doi.org/10.1109/HPCS.2017.87

59. Benedicenti, L., Cotugno, F., Ciancarini, P., Messina, A., Pedrycz, W., Sillitti, A., Succi, G. Applying scrum to the army: a case study. *International Conference on Software Engineering Companion IEEE*, 2016. p.p. 725-727. https://doi.org/10.1145/2889160.2892652

60. Bertling M., Caroli H., Dannapfel M., Burggraf P. The minimal viable production system (mvps) – an approach for agile (automotive) factory planning in a disruptive environment. *Advances in Production Research. Springer International Publishing.* 2019. pp. 24–33, https://doi.org/10.1007/978-3-030-03451-1_3

61. Boral, S. Domain V: Adaptive Planning. *Ace the PMI-ACP® exam. Apress, Berkeley*, 2016. CA. https://doi.org/10.1007/978-1-4842-2526-4_6

62. Carbone, R., Barone, S., Barbareschi & M., Casola, V. Scrum for Safety: Agile Development in Safety-Critical Software Systems. Quality of Information and Communications Technology. *Communications in Computer and Information Science*, 2021. P. 1439. https://doi.org/10.1007/978-3-030-85347-1_10

63. Casola, V., De Benedictis, A., Rak, M., Villano, U. A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach. *Journal of Systems and Software*, 2020. P. 163. https://doi.org/10.1016/j.jss.2020.110537

64. Cawley, O., Wang, X., Richardson, I. Lean/agile software development methodologies in regulated environments-state of the art. *Lean Enterprise Software and Systems. LESS 2010. Lecture Notes in Business Information Processing*, 2010. P. 65. https://doi.org/10.1007/978-3-642-16416-3_4

65. Cole R., Scotcher E. Brilliant Agile Project Management: A Practical Guide to Using Agile, Scrum and Kanban. Edinburg: Pearson, 2015. 187 p.

66. Cotugno, F.R., Messina, A. Adapting SCRUM to the Italian Army: Methods and (Open) Tools. Open Source Software: Mobile Open Source Technologies. *OSS 2014. IFIP Advances in Information and Communication Technology*, 2014. P. 427. https://doi.org/10.1007/978-3-642-55128-4_7

67. De Sá, F.R., Vieira, R.G., da Cunha, A.M. Lessons Learned from the Agile Transformation of an Aeronautics Computing Center. Agile Methods. *WBMA 2019. Communications in Computer and Information Science*, 2019. P. 1106. https://doi.org/10.1007/978-3-030-36701-5_7

68. Dur´an, M., Ju´arez-Ram´ırez, R., Jim´enez, S., Tona, C. User story estimation based on the complexity decomposition using bayesian networks. *Programming and Computer Software* 46, 2020. p.p. 569–583 https://doi.org/10.1134/S0361768820080095

69. Dymova, H., Larchenko, O.: Development of a computer program for solving network optimization problems. *Computer-integrated technologies: education, science, production.* 41, 2020. p.p. 143–151 https://doi.org/10.36910/6775-2524-0560-2020-41-23

70. Edison H., Wang X. and Conboy K. (2022) Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. Transactions on Software Engineering, 48(8), 2020. p.p. 2709-2731. https://doi.org/10.1109/TSE.2021.3069039

71. Gozhyj A., Kalinina I., Gozhyj V., Vysotska V., Web Service Interaction Modeling with Colored Petri Nets. *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Metz, France, 2019, pp. 319-323, doi: 10.1109/IDAACS.2019.8924400.

72. Gozhyj, A., Kalinina, I., Gozhyj, V., Danilov, V. Approach for Modeling Search Web-Services Based on Color Petri Nets. *Data Stream Mining & Processing. DSMP 2020. Communications in Computer and Information Science*, vol 1158, 2020. Springer, Cham, https://doi.org/10.1007/978-3-030-61656-4_35

73. Guidance on the use of AGILE practices in the development of medical device software (2012). Retrieved from: https://webstore.ansi.org/standards/aami/aamitir452012r2018

74. Hajou, A., Batenburg, R., Jansen, S. How the pharmaceutical industry and agile software development methods conflict: A systematic literature review. *International*

*Conference on Computational Science and Its Applications IEEE*, 2014. p.p. 40-48, https://doi.org/10.1109/ICCSA.2014.19

75. Hallmann, D. ”i don’t understand!”: Toward a model to evaluate the role of user story quality. *Agile Processes in Software Engineering and Extreme Programming*. 2020. pp. 103–112. Springer International Publishing https://doi.org/10.1007/978-3-030-49392-9_7

76. Hanssen, G., Stålhane, T., & Myklebust, T. SafeScrum – Agile Development of Safety-Critical Software. New York: Springer. 2020.

77. Hovorushchenko T., Herts A.; Hnatchuk Y. Concept of Intelligent Decision Support System in the Legal Regulation of the Surrogate Motherhood. *International Workshop on Informatics & Data-Driven Medicine*, 2019, pp: 57-68.

78. Islam G., Stoner T. A case study of agile software development for safety-Critical systems projects. *Reliability Engineering & System Safety*. Vol. 200. 2020. https://doi.org/10.1016/j.ress.2020.106954

79. ISO/IEC 12207:2008 «Systems and software engineering — Software life cycle processes» [Електронний ресурс] – URL: https://standards.ieee.org/ieee/12207/5672/

80. Jain P., Sharma A., Ahuja L. The Impact of Agile Software Development Process on the Quality of Software Product. *International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO).* Noida, India, 2018, pp. 812-815, https://doi.org/10.1109/ICRITO.2018.8748529

81. Jansen U., Schulz W. FlowChart Tool for Decision Making in Interdisciplinary Research Cooperation. Digital Human Modeling. Applications in Health, Safety, Ergonomics, and Risk Management: Health and Safety. *Lecture Notes in Computer Science*, 2017. vol 10287. Springer, Cham. https://doi.org/10.1007/978-3-319-58466-9_24

82. Karras, O., Kl¨under, J., Schneider, K.: Is task board customization beneficial? *International Conference on Product-Focused Software Process Improvement*. 2017. pp. 3–18. Springer https://doi.org/10.1007/978-3-319-69926-4_1

83. Khmel, M., Prydatko, O., Popovych, V., Tkachenko, T., Kovalchuk, V.: Students r&d projects as a tool for achieving program competencies. *Information and communication technologies in modern education: experience, problems, prospects*. 2021.

84. Kordunova Y., Prydatko O., Smotr O., Golovatyi R. Expert Decision Support System Modeling in Lifecycle Management of Specialized Software. *Lecture Notes on Data Engineering and Communications Technologies*, *Springer*, Switzerland. Vol. 149, 2022, pp. 367-383, https://doi.org/10.1007/978-3-031-16203-9_22

85. Kordunova Yu., Prydatko O., Smotr O., Kokotko I. The network graph traversal method for solving the problem of short-term planning of safety-oriented services development. Monografia powstała w ramach Projektu dofinansowanego przez Ministra Edukacji i Nauki ze środków budżetu państwa w ramach programu „Doskonała Nauka", Warszawa 2022. p. 172–181.

86. Kovalchuk O., Zachko O. and Kobylkin D. Criteria for intellectual forming a project teams in safety oriented system. *IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT)*, Lviv, Ukraine, 2022, pp. 430-433

87. Kovalchuk O., Kobylkin D. and Zachko O. HR Decision-Making Support System Based On The CBR Method. *IEEE 18th International Conference on Computer Science and Information Technologies (CSIT)*, Lviv, Ukraine, 2023, pp. 1-4, doi: 10.1109/CSIT61576.2023.10324169.

88. Kuhrmann M. et al., Hybrid Software Development Approaches in Practice: *A European Perspective. IEEE Software.* 2019. vol. 36, no. 4, pp. 20-31, https://doi.org/10.1109/MS.2018.110161245

89. Kula E., Greuter E., Deursen A., Gousios G. Factors Affecting On-Time Delivery in Large-Scale Agile Software Development. *IEEE Transactions on Software Engineering*, vol. 48, no. 9. 2022. pp. 3573-3592. https://doi.org/10.1109/TSE.2021.3101192.

90. Lamsellak H, Belkasm M. G. Global software development agile planning model:challenges and current trends. Indonesian Journal of Electrical Engineering and Computer Science, Vol. 32, No. 3. 2023. pp. 1774-1784

91. Lamsellak H., Khalil A., Belkasmi M.G., Saber M. Agile Practices in Iteration Planning Process of Global Software Development. International Conference on Advanced Intelligent Systems for Sustainable Development. AI2SD 2022. Lecture Notes in Networks and Systems, vol 712, 2023. Springer, Cham, https://doi.org/10.1007/978-3-031-35251-5_29

92. Lenarduzzi V., Lunesu I., Matta M., Taibi D. Functional Size Measures and Effort Estimation in Agile Development: A Replicated Study. Agile Processes in Software Engineering and Extreme Programming. XP 2015. *Lecture Notes in Business Information Processing*, 2015. vol 212. https://doi.org/10.1007/978-3-319-18612-2_9

93. Liaskovska, S., Martyn, Y., Malets, I., Prydatko, O.: Information technology of process modeling in the multiparameter systems. *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP),* Lviv, Ukraine, 2018, pp. 177–182 (08 2018). https://doi.org/10.1109/DSMP.2018.8478498

94. Malets I., Prydatko O., Popovych V., Dominik A. Interactive Computer Simulators in Rescuer Training and Research of their Optimal Use Indicator. *2018 IEEE Second Conference on Data Stream Mining & Processing*. Lviv. №2. 2018. p.p. 558-562.

95. Martyn, Y., Smotr, O., Burak, N., Prydatko, O., Malets, I.: Software for shelter's fire safety and comfort levels evaluation. *Communications in Computer and Information Science*. 2020. pp. 457–469. Springer International Publishing https://doi.org/10.1007/978-3-030-61656-4_31

96. Martyn Ye., Liaskovska S., Gregus M., Izonin I., Velyka O. Optimization of Technological's Processes Industry 4.0 Parameters for Details *Manufacturing via Stamping: Rules of Queuing Systems. Procedia Computer Science*. 2021. Vol. 191, 290-295

97. McCaffery, F., Trektere, K., Ozcan-Top, O. Agile – Is it Suitable for Medical Device Software Development? *Software Process Improvement and Capability Determination*, 2016. P. 609. https://doi.org/10.1007/978-3-319-38980-6_30

98. Meier, A., Ivarsson, J.C.: Agile software development and service science. GSTF *Journal on Computing (JoC)* 3(3), 2013. p.p. 1–5 https://doi.org/10.7603/s40601-013-0029-6

99. Messina, A., Fiore, F., Ruggiero, M., Ciancarini, P., Russo, D. (2016). A new agile paradigm for mission-critical software development. *CrossTalk*. 29, 2016. p.p. 25-30.

100. Notander, J. P., Runeson, P., Höst, M. A model-based framework for flexible safety-critical software development: a design study. *Proceedings of the 28th Annual ACM Symposium on Applied Computing.* 2013. p.p. 1137-1144, https://doi.org/10.1145/2480362.2480575

101. Paez N., Fontdevila D., Oliveros A. On the Influence of Agile in the Usage of Software Development Practices. I*EEE Congreso Bienal de Argentina (ARGENCON),* Resistencia Argentina. 2020. p.p. 1-7, https://doi.org/10.1109/ARGENCON49523.2020.9505407.

102. Papadopoulos G. Moving from traditional to agile software development methodologies also on large, distributed projects. *Procedia – Social and Behavioral Sciences.* 2015. № 175. pp. 455 – 463. https://doi.org/10.1016/j.sbspro.2015.01.1223

103. Prydatko, O., Popovych, V., Malets, I., Solotvinskyi, I.: Algorithm of rescue units logistic support planning in the process of regional life safety systems development. *MATEC Web of Conferences*. 294, 04002 (01 2019). https://doi.org/10.1051/matecconf/201929404002

104. Rajkumar S., Pradeep K., Partha P. R., Umapada P. Modeling local and global behavior for trajectory classification using graph-based algorithm. *Pattern Recognition Letters*. 2021, volume 150, pp 280-288

105. Ribeiro S., Schmitz E., Alencar A., Silva M. Literature Review on the Theory of Constraints Applied in the Software Development Process. *IEEE Latin America Transactions.* 2018. vol. 16, no. 11, pp. 2747-2756, https://doi.org/10.1109/TLA.2018.8795116

106. Sachdeva V. Requirements Prioritization in Agile: Use of Planning Poker for Maximizing Return on Investment. Information Technology - New Generations. *Advances in Intelligent Systems and Computing*, 2017. vol 558. Springer, Cham. https://doi.org/10.1007/978-3-319-54978-1_53

107. Schwaber, K., & Sutherland, J. (2020). The scrum guide (МетодР16)

108. Seidykh, O., Chobanu, V.: Optomozation of the network graphics of the complex of works. *Modern engineering and innovative technologi.* 2018. 1(3), p.p. 61–67. https://doi.org/10.30890/2567-5273.2018-03-01-009

109. Singh B. and Gautam S. The Impact of Software Development Process on Software Quality: A Review. *8th International Conference on Computational Intelligence and Communication Networks (CICN).* Tehri, India, 2016. pp. 666-672, https://doi.org/10.1109/CICN.2016.137

110. Smith J., Bradbury J., Hayes W., Deadrick W. Agile Approach to Assuring the Safety-Critical Embedded Software for NASA's Orion Spacecraft. *IEEE Aerospace Conference, Big Sky*, 2019. p.p. 1-10, https://doi.org/10.1109/AERO.2019.8742095.

111. Steghöfer, JP., Knauss, E., Horkoff, J., Wohlrab, R. Challenges of Scaled Agile for Safety-Critical Systems. In: Franch, X., Männistö, T., Martínez-Fernández, S. (eds) Product-Focused Software Process Improvement. PROFES 2019. *Lecture Notes in Computer Science*, 2019. P. 11915. Springer, Cham. https://doi.org/10.1007/978-3-030-35333-9_26

112. Stellman A., Greene J. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. 1st Edition, USA: O'Reilly Media, 2013. 420 c.

113. Stioca M., Ghlic-Micu B., Mircea M., Uscatu C. Analyzing Agile Development – from Waterfall Style to Scrumban. *Informatica Economică*. 2016. №4. С. 5–14. https://doi.org/10.12948/issn14531305/20.4.2016.01

114. Thesing T., Feldmann C., Burchardt M. Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project. *Procedia Computer Science*. 2021. № 181. pp. 746–756.

115. Tordesillas, J., Lopez, B.T., Carter, J., Ware, J., How, J.P.: Real-time planning with multi-fidelity models for agile flights in unknown environments. 2019. pp. 725–731 https://doi.org/10.1109/ICRA.2019.8794248

116. Uzun, I., Lobachev, I., Gall, L., Kharchenko, V.: Agile Architectural Model for Development of Time-Series Forecasting as a Service Applications. 2021. pp. 128–147. Springer International Publishing https://doi.org/10.1007/978-3-030-82014-5_9

117. Velykodniy, S., Burlachenko, Z., Zaitseva-Velykodna, S.: Architecture development of software for managing network planning of software project reengineering. *Innovative technologies and scientific solutions for industries*. 2019. № 2(8), p.p. 25–35 https://doi.org/10.30837/2522-9818.2019.8.025

118. Wang Linlin, Fang Xiaoyu, Hong Tao, Liu Chang, Liu Shilan Image Recognition Based on the Depth-Wise Separable Convolution and Softpool. *International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, 2022. pp.147-152

119. Wang H., Ma Z. Application and Improvement of Agile Development in Intelligent Health Hut Software Project. *International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS)*, 2023. p.p. 14-18, https://doi.org/10.1109/ICMSS56787.2023.10118316.

120. Zachko O. B., Kobylkin D. S. Structural Models of Safety-Oriented Management of Infrastructure Projects Decomposition. *International Scientific and Technical Conference on Computer Sciences and Information Technologies*. 2020. № 2. P. 131–134.

121. Zachko O. B., Kovalchuk O. I. Models of the life cycle of forming project teams in a security-oriented system. *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, 2020. Vol. 1, P. 235-239.