

## УПРАВЛІННЯ ПРОЦЕСАМИ У РОЗПОДІЛЕНИХ ПРОЄКТНИХ КОМАНДАХ

Роман Васьків<sup>1</sup>, Наталія Веретеннікова<sup>2</sup>

<sup>1,2</sup> Національний університет “Львівська політехніка”,  
кафедра інформаційних систем та мереж, Львів, Україна

<sup>1</sup> E-mail: roman.i.vaskiv@lpnu.ua, ORCID: 0000-0002-8549-5035

<sup>2</sup> E-mail: nataliia.v.veretennikova@lpnu.ua, ORCID: 0000-0001-9564-4084

© Васьків Р., Веретеннікова Н., 2024

У роботі досліджено підходи до управління процесами координації та розподілу завдань у розподілених проєктних командах в ІТ-галузі, які працюють у географічно розосереджених Agile-середовищах. Основну увагу приділено розробленню моделі відбору членів команди, яка враховує досвід, продуктивність і географічне розташування членів розподілених команд для ефективного виконання завдань. Запропонована функціональна модель надає можливість врахувати основні фактори та залежності, що впливають на процес прийняття рішень у розподілених командах, зокрема функціональність, часову доступність і географічну відповідність. Це сприяє мінімізації ризиків і підвищенню ефективності управління проєктами, що особливо важливо в умовах швидко змінюваного ІТ-ринку. Проаналізовано обмеження моделі та запропоновано шляхи подальших розвідок.

**Ключові слова:** розподілені проєктні команди, Agile, розподілена розроблення програмного забезпечення, управління проєктними процесами.

### Формулювання проблеми

У сучасному ІТ-середовищі розподілені команди набувають усе більшої популярності завдяки глобалізації ринку праці та розвитку технологій дистанційної роботи, а також процесам, які стимулюють компанії частіше впроваджувати віддалені технології виробничої взаємодії. Це своєю чергою дає змогу компаніям швидко реагувати на зміни ринку, залучаючи кваліфікованих фахівців у різних географічних локаціях. Завдяки цьому фірми, компанії та виробництва можуть працювати цілодобово, використовуючи синхронізацію виконання завдань у різних часових поясах для безперервності потоку їхнього виконання.

Сучасні підходи до розроблення програмного забезпечення орієнтовані на швидку доставку якісних продуктів з мінімальними витратами. Agile-методології, орієнтовані на задоволення потреб замовників, стали ефективним інструментом досягнення цих цілей, роблячи акцент на співпраці між учасниками команди, швидкій адаптації до змін і націленості на створення якісних високофункціональних продуктів.

### Аналіз останніх досліджень та публікацій

Згідно з дослідженням Інституту з управління проєктами (Project Management Institute, PMI), тенденції у сферах віддаленої роботи та розподілених команд продовжують зростати. Гібридні підходи до управління проєктами, які поєднують елементи Agile і Predictive методологій, значно

збільшилися у використанні, зрісши від 20 % 2020 року до 31,5 % 2023 року. Це вказує на зміщення трендів до комбінування цих підходів для кращого управління розподіленими командами в різних галузях [1].

Однак упровадження зазначених методологій у розподілених умовах, коли команди розміщені в різних частинах світу, створює нові виклики, які впливають на ефективність управління проєктами.

Управління процесами формування та динамічної актуалізації переліку завдань, необхідних для розроблення продукту, що ґрунтується на аналізі ринкових вимог, зворотного зв'язку користувачів і можливостей виробництва (беклог), у розподілених проєктних командах супроводжується специфічними викликами, пов'язаними з координацією та ефективністю виконання [2]. Географічна розподіленість учасників ускладнює комунікацію та координацію завдань, часова розподіленість може призвести до затримок у виконанні робіт через різницю у часових поясах, а функціональна розподіленість вимагає точного розподілу ролей та обов'язків серед учасників команди. Усі ці фактори створюють додаткові перешкоди і вимагають розроблення нових методів та моделей для оптимального розподілу завдань і підвищення загальної ефективності управління процесами виконання таких проєктів.

Розподілене розроблення програмного забезпечення (Distributed Software Development, DSD) охоплює взаємодію членів команди, які фізично розташовані у різних локаціях, що може залучати і різні регіони в межах однієї країни, і різні країни та континенти. Цей підхід відкриває нові можливості для удосконалення процесів розроблення, зокрема скорочення часу розроблення, зменшення витрат і підвищення якості кінцевого продукту завдяки залученню найефективніших фахівців заданих профілів та кваліфікації з різних регіонів. Проте впровадження розподілених Agile-середовищ (Distributed Agile Software Development, DASD) супроводжується серйозними викликами, як-от управління міжкомандною взаємодією, забезпечення ефективної комунікації та координації, а також подолання культурних і мовних бар'єрів [3].

Однією з ключових проблем у таких умовах є розподіл завдань серед членів команди. Непрозорий і недостатньо виважений та обґрунтований розподіл може призвести до зниження якості програмного продукту, падіння продуктивності праці та мотивації команди. Хоча належне планування проєкту є необхідною передумовою для успішної розроблення, на практиці доволі багато проєктів зазнають невдач через неефективну організацію виконання завдань і управління проєктними командами.

У традиційних Agile-підходах розподіл завдань ґрунтується на ідентифікації історій користувачів (User Stories), де кожна історія розбивається на завдання, які повинні реалізовувати нові функціональні можливості. Однак якщо члени команди розподілені географічно, прийняття рішень щодо розподілу завдань ускладнюється через відсутність можливості проведення прямих зустрічей. Це призводить до ускладнень у координації, необхідної для узгодження завдань серед членів команди.

Розподілена розроблення проєктів є порівняно новою парадигмою, яка, окрім переваг, супроводжується певними ризиками. Зокрема, підвищена складність розподілу завдань виникає через недостатнє врахування географічного розташування команд, різниці в часових поясах і можливих культурних конфліктів. Ефективний розподіл завдань стає критично важливим для прийняття оптимальних рішень, що впливають на успішність проєкту та допомагають мінімізувати ризики.

Ефективне управління процесами в розподілених проєктних командах може бути вдосконалене через впровадження базових принципів розподілених інформаційних систем. Використання проміжного програмного забезпечення (middleware) в інформаційних системах забезпечує взаємодію між різнорідними компонентами, що сприяє узгодженості процесів [4]. Аналогічно для розподілених команд проміжне програмне забезпечення (middleware) дозволить налаштувати інформаційний супровід, що покращує комунікацію між віддаленими членами команди, мінімізуючи

труднощі, пов'язані з різницею в часових поясах і географічною розподіленістю, та сприяючи підвищенню ефективності проєктів.

Ефективне управління розподіленими проєктними командами, особливо в умовах Agile-методологій, є складним завданням, яке вимагає інтеграції нових підходів до координації та розподілу завдань. Низка досліджень підкреслює необхідність удосконалення процесів, технологій та методів управління для забезпечення успішної роботи розподілених команд.

Форсгрєн Н., Хамбл Дж. і Кім Г. підкреслюють важливість інтеграції практик Lean та DevOps для підвищення продуктивності програмних команд. Їхнє дослідження показує, що такі підходи можуть значно покращити ефективність роботи розподілених команд завдяки оптимізації швидкості та якості доставки продукту. Вони наголошують на необхідності адаптації комунікаційних та управлінських методів, що є критично важливим для розподілених команд [5].

Подібно Ламерсдорф А. та ін. [6] повідомляють, що практики вважають, що їхні проєкти з розподіленими командами зазнали невдачі через те, що розподіл завдань був занадто зосереджений на одному критерії (зазвичай на вартості робочої сили), ігноруючи інші важливі фактори, як-от: культурні відмінності, індивідуальний досвід, експертиза, близькість до клієнтів і взаємозв'язок між завданнями. Незважаючи на значущість цієї проблеми, досліджень, присвячених факторам, що впливають на розподіл завдань у проєктах GSD, усе ще мало [7].

Сіман Фільо, М., Піньейро, П. Р. та інші автори аналізують різні підходи до розподілу завдань у розподілених командах, зокрема багатокритеріальні моделі, які дають змогу оптимізувати процеси координації та понизити ризики, пов'язані з комунікаційними бар'єрами та різницею у часових зонах. Вони підкреслюють, що врахування географічних та часових особливостей учасників команди є одним з ключів успішного управління проєктами [8].

Перейра Л. та інші, аналізуючи публікації, в яких подано результати дослідження віртуальних команд, підкреслюють важливість таких аспектів, як-от динаміка команд, використання комунікаційних технологій. Вони акцентують увагу на викликах, з якими стикаються при управлінні проєктами в умовах проєктів, що реалізуються віртуальними командами, та на необхідності адаптації стратегій управління до нових умов [9].

Аслам В., Іджаз Ф. пропонують оригінальну модель для розподілу завдань у розподілених Agile-командах, що базується на математичних методах оцінки. Це дослідження є важливим для розуміння того, як математичні моделі можуть використовуватися в процесах управління завданнями, даючи змогу підвищити ефективність координації між членами команди та зменшити ризики, пов'язані з комунікаційними перешкодами [10].

Управління проєктними процесами у розподілених командах часто передбачають складні взаємодії між експертами, ролями, командами та завданнями. У цьому контексті особливо важливою є координація на трьох рівнях: індивідуальному, груповому та програмному, завдяки чому можна ефективно організувати розподіл завдань у IT-проєктах із розподіленими командами. Важливо також враховувати різні типи комунікацій, зокрема зустрічі у межах Agile та глобальної розроблення, а сучасні комунікаційні технології можуть слугувати засобами для ефективної взаємної адаптації та координації в таких командах [11]. Як зазначають Ван Стін і Таненбаум (2024), ключовим викликом у розподілених системах є забезпечення цілісної інтеграції компонентів через координацію процесів і комунікацію між командами, що працюють у різних географічних зонах, а також адаптація цих процесів до вимог динамічного IT-середовища, де важлива стабільність і ефективність комунікаційних протоколів на всіх рівнях [4]. Важливими факторами є врахування часових зон та синхронізації комунікації, що сприяє зниженню ризиків та покращенню інтеграції у розподілених IT-проєктах [12–14].

### Формулювання цілі статті

У роботі подано розроблення та створення функціональної моделі управління процесами в розподілених проєктних командах, яка враховує ключові фактори: функціональність, часову узгодже-

ність і географічну віддаленість учасників. Статтю сфокусовано на підвищенні ефективності командної роботи через оптимізацію розподілу завдань та координації в умовах розподілених Agile-команд, а також пропонує рішення для подолання викликів і мінімізації ризиків, що виникають у розподілених командах, для забезпечення підвищення загальної продуктивності та успішної реалізації проектів.

### Виклад основного матеріалу

Процес управління розподіленими командами в IT-проектах стає все більш складним, особливо в умовах інтеграції світових ринків та все більш поширеного використання віддалених режимів роботи. Формування ефективної команди, яка відповідає вимогам проекту, є ключовим завданням для забезпечення успішної його реалізації. Однак традиційні методи управління та підбору персоналу часто не враховують такі важливі аспекти, як-от: функціональність, часова доступність і географічне розташування членів команди.

Для вирішення цієї проблеми була розроблена концептуальна модель інформаційної системи, в якій автоматизуються процеси відбору членів команди на основі моделей, які враховують три базові характеристики розподіленості. Крім того, запропонована інформаційна система забезпечує ефективну комунікацію між членами команди та замовником продукту на всіх етапах розроблення, від задач проектування до системної інтеграції та тестування.

Функціональна блок-схема процесів візуалізує процедуру управління проектом, починаючи з початкового планування і закінчуючи доставкою окремих підсистем, що становлять цілісну систему, замовнику. З її допомогою подана ілюстрація взаємозв'язків на різних етапах проекту, зокрема йдеться про відбір команди, управління завданнями, комунікацію між учасниками та інтеграцію компонентів (рис. 1).

Таблиця 1

### Основні терміни, позначення та їхнє подання

Залежність сутності (Entity Dependency – ED)	Залежність між сутностями в інформаційній системі, де одна сутність впливає на функціональний стан іншої, забезпечуючи цілісність і узгодженість процесів. Інформаційний обмін відбувається між сутностями, а не між процесами.
Залежність діяльності (Activity Dependency – AD)	Залежність між процесами, де один процес повинен завершитися до початку іншого.
Послідовна залежність (Sequential Dependency – SD)	Послідовна залежність між процесами, що виконуються в певному порядку.
Функціональна залежність (Functional Dependency – FD)	Залежність, що визначає відповідність завдань функціональним вимогам.
Комунікаційний процес (Communication Process – CP)	Процес комунікації між учасниками проекту, зокрема між Development Team (командою проекту) і Product Owner (власником продукту).
Оцінка функціональності (Functionality Score – FS <sub>i</sub> )	Оцінка відповідності кандидата функціональним вимогам.
Оцінка тимчасової узгодженості (Temporal Alignment Score – TAS <sub>i</sub> )	Оцінка відповідності кандидата часовим параметрам (робочим годинам).
Оцінка географічної відповідності (Geographical Suitability Score – GSS <sub>i</sub> )	Оцінка відповідності кандидата географічному розташуванню.
Умова для ухвалення рішення (Decision Condition – DC)	Умови для ухвалення рішення щодо відповідності кандидата.
Історія користувача (User Story – US)	Стислий опис кінцевого результату для конкретного користувача, що є запрошенням до обговорення деталей. [3, PMBOK].
Беклог (Backlog)	Впорядкований список завдань або елементів роботи, які мають бути виконані в рамках проекту [3, PMBOK].
Власник продукту (Product Owner)	Особа, відповідальна за максимізацію цінності продукту та ухвалення рішень стосовно кінцевого продукту [3, PMBOK].

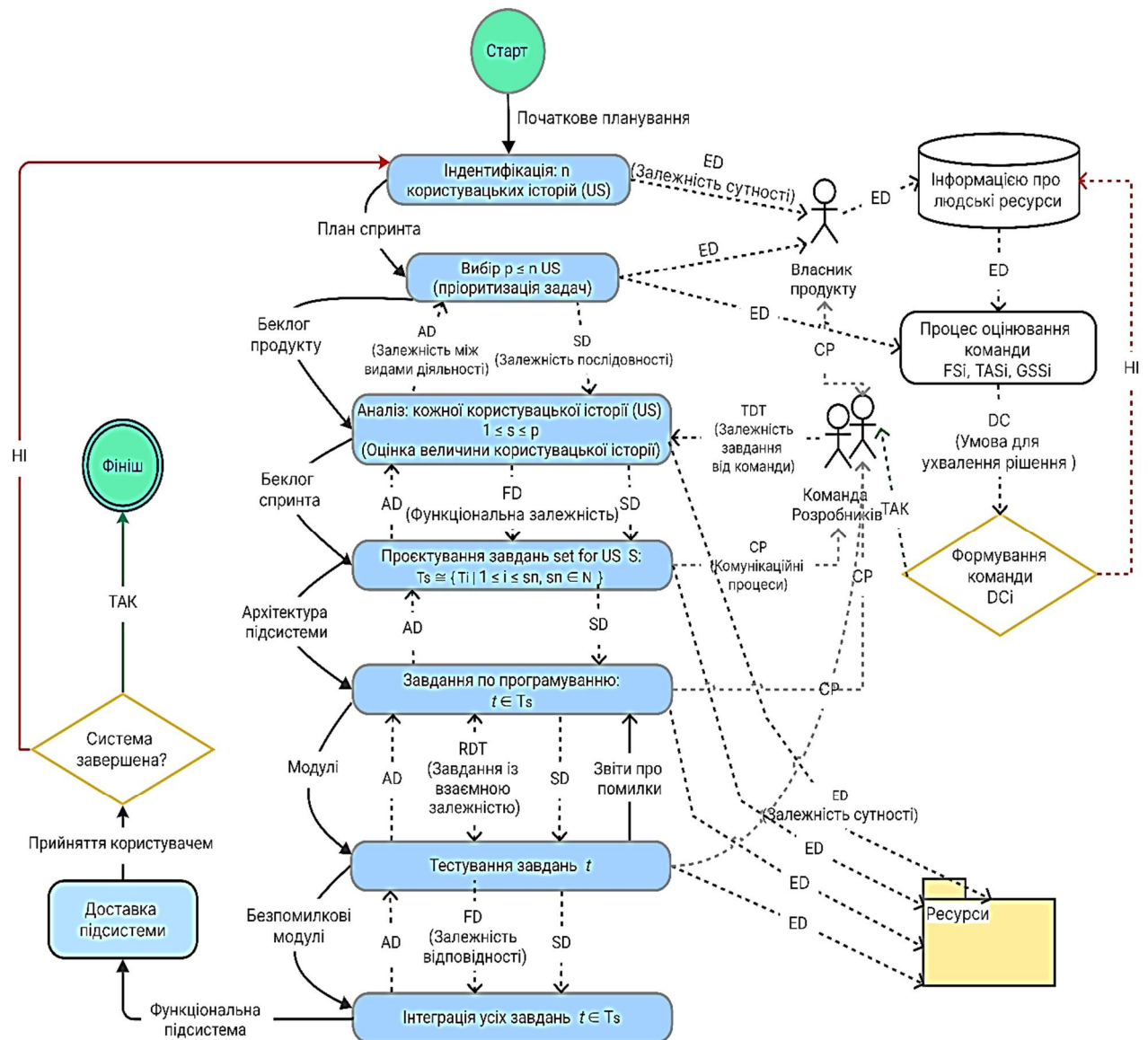


Рис. 1. Модель процесу ІТ-розроблення з процедурами оптимального вибору розподіленої проектної команди на основі функціональної, часової та географічної сумісності, а також оцінки його ефективності для зниження ризиків і зменшення вартості проекту

Функціональна модель, подана діаграмою, зорієнтована на процедури автоматизованого відбору кандидатів для розподілених команд на основі трьох ключових параметрів: функціональність, часова доступність та географічна відповідність. Використовуючи наведену модель, керівники надають додаткові інструменти та можливості щодо оптимізації процесів формування команди, забезпечуючи водночас мінімізацію ризиків та максимізацію ефективності роботи над проектом.

У запропонованій блок-схемі відображено повний цикл управління проектом, охоплюючи відбір команди, виконання завдань, координацію між учасниками та завершення проекту з передаванням готової системи або її компонентів замовникові.

Розглянемо більш детально запропоновану функціональну модель.

### 1. Ініціація проекту (Project Initiation)

Ініціація проекту (Project Initiation) є стартовою точкою розроблення ІТ-проекту та ключовим етапом запуску всіх процесів управління проектом. На цьому етапі визначають основні передумови,

встановлюють цілі, формулюють очікування та обмеження, що слугують базою для подальших етапів планування та реалізації. Етап ініціації тісно пов'язаний із усіма подальшими процесами, оскільки він закладає фундаментальні передумови, на яких базуватиметься весь проект.

Після ініціації проект переходить у фазу Початкового планування (Initial Planning), яка конкретизує базові аспекти проекту, як-от: ресурси, обмеження та загальний напрямок розвитку. Хоча цілі були встановлені на етапі ініціації, на цьому етапі вони деталізуються і формалізуються в межах проектної документації. Цей етап тісно пов'язаний із подальшими процесами, зокрема з ідентифікацією User Stories (USs), що є основою для створення – впорядкованого списку завдань, виконання яких є обов'язковим для створення якісного, високотехнологічного та конкурентно спроможного (беклогу) продукту. На цьому етапі відбувається активна взаємодія з Власником продукту (Product Owner), який, використовуючи принципи залежностей сутностей (Entity Dependency – ED), визначає пріоритети й очікування для наступних ітерацій. Взаємодія між різними сутностями проекту забезпечує узгодженість цілей та вимог на всіх етапах його реалізації.

Власник продукту може отримати доступ до бази даних з Інформацією про людські ресурси (Human Resources Information), яка містить детальну інформацію про потенційних учасників команди і внутрішніх, і зовнішніх кандидатів. У цій базі даних відображені такі параметри потенційних виконавців, як-от: досвід, навички, предметна область, географічне розташування та доступність, зокрема часові пояси. Ця інформація слугує допоміжним інструментом при формуванні команди з використанням моделі, яка враховує розподіл завдань між учасниками на основі їхніх компетенцій, спеціалізацій, географічного розташування, часового поясу та функціональних ролей. Такий підхід дає змогу сформувати високо-фахову команду для ефективного виконання проекту.

Отже, цей етап виступає важливою відправною точкою, що приводить у дію всі подальші етапи проекту, які відображені блок-схемою, забезпечуючи структуроване та ефективне управління процесами розроблення програмного забезпечення.

## 2. План спринту (Sprint Plan)

Після завершення початкового етапу (Start), проект переходить у фазу План спринту (Sprint Plan), яка визначає, як саме і які завдання будуть виконувати впродовж поточного спринту. Ця фаза характеризується станом:

$$\text{Selection } p \leq n \text{ US (prioritization)}, \quad (1)$$

де відбувається вибір і пріоритизація завдань (User Stories, US) для виконання в поточному спринті.

- **Selection** процес відбору завдань або користувачьких історій (User Stories, US).
- **$p \leq n$  US** вказує на те, що з усієї наявної кількості користувачьких історій ( $n$ ), для реалізації обираються лише  $p$  найбільш пріоритетних завдань, де  $p$  є кількістю завдань, які команда може виконати у межах одного спринту.
- **Prioritization** – підкреслює, що процес відбору базується на пріоритизації завдань, тобто на виборі найбільш важливих та необхідних для досягнення цілей спринту.

На цьому етапі суттєву роль відіграє Власник продукту (Product Owner) – особа, що є відповідальною за максимізацію цінності створюваного продукту, який через залежність ED (Entity Dependency) визначає перелік пріоритетних задач. Водночас відбувається взаємодія з Team Evaluation Process для визначення найкращих претендентів у члени команди. Процес оцінювання здійснюється на основі трьох ключових параметрів: функціональність, часова доступність та географічна відповідність.

Модель оцінювання кандидатів: Functionality Score (FS<sub>i</sub>).

Цей показник оцінює відповідність кандидата вимогам задачі на основі його знань, навичок і досвіду. До уваги можуть братися фактори роботи з іншими проектами в цій самій предметній ділянці, експертності у певній технології.

$$FS_i = \sum_{j=1}^n W_j \times C_{ij}, \quad (2)$$

де  $W_j$  – вага критерію знань і досвіду, а  $C_{ij}$  – рівень відповідності кандидата  $i$  критерію  $j$ .

Подамо послідовність етапів більш деталізовано, а саме:

**Визначення критеріїв → Призначення ваг для критеріїв → Оцінка кандидатів → Розрахунок оцінки функціональності → Вибір найкращого кандидата**

Таблиця 2

**Етапи та опис процесу відбору претендентів для проєктної команди**

Етап	Опис
Визначення критеріїв	Визначаються ключові критерії для кожної задачі або набору задач у проєкті. Вони передбачають: знання конкретних технологій, досвід роботи з певними інструментами, знання методологій, навички роботи в команді.
Призначення ваг для критеріїв	Для кожного критерію призначають вагу $W_j$ , яка відображає його відносну важливість у контексті задачі. Вага допомагає визначити пріоритетність кожного критерію.
Оцінка кандидатів	Кожного кандидата оцінюють на відповідність критеріям. Оцінка $C_{ij}$ базується на попередньо зібраних даних, результатах тестових завдань або іншій релевантній інформації.
Розрахунок оцінки функціональності	Розраховується загальна оцінка функціональності (Functionality Score) $FS_i$ для кожного кандидата. Цей показник визначається як сума добутків ваги кожного критерію на оцінку відповідності кандидата.
Вибір найкращого кандидата	Кандидатів із найвищою функціональною оцінкою (Functionality Score) відбирають для наступного етапу відбору, оскільки вони є найбільше підходять для розв'язання задач.

Реалізація цих етапів забезпечує об'єктивний та ефективний процес відбору, що є критично важливим для формування оптимальної команди та досягнення успішних результатів у проєкті.

**TAS<sub>i</sub>** (Temporal Alignment Score) – показник оцінює відповідність кандидата часовим параметрам, тобто його доступності у відповідні години роботи:

$$TAS_i = \frac{P_{i1} \cap P_{i2} \cap \dots \cap P_{in}}{T_{total}} \times (1 - L_i), \quad (3)$$

де  $P_{i1}, P_{i2}, \dots, P_{in}$  – доступні робочі години кандидата, а  $T_{total}$  – загальна тривалість робочого часу.

$T_{total}$  – загальна тривалість робочого часу (може бути визначена для проєкту, наприклад, 8 годин на день або 40 годин на тиждень).

$L_i$  – завантаження кандидата іншими проєктами (виражене у відсотках, наприклад, 0,9 для 90 % зайнятості). Отже, коефіцієнт  $(1 - L_i)$  відображає частку часу, яку кандидат може виділити для нового проєкту. Якщо кандидат завантажений на 90 % ( $L_i = 0,9$ ). Він має лише 10 % доступного часу для нового проєкту.

Якщо  $TAS_i$  наближається до 1. Це означає, що кандидат ідеально підходить: його робочий час збігається з іншими членами команди, і він не перевантажений іншими проєктами.

Якщо  $TAS_i$  низький. Це означає, що кандидат або зайнятий іншими проєктами, або його робочий час не відповідає часу інших членів команди.

Вказана оцінка дає змогу знайти найкращих кандидатів, які працюють у відповідних часових зонах і мають достатньо вільного часу для обраного проєкту.

**Оцінка географічної відповідності (Geographical Suitability Score – GSS<sub>i</sub>)** – показник оцінює географічну відповідність кандидата, враховуючи близькість його розташування до інших членів команди та вплив на ефективність комунікації:

$$GSS_i = \frac{1}{D_{ij}} \times T_{ij}, \quad (4)$$

де  $D_{ij}$  – географічна відстань між членами команди, а  $T_{ij}$  – коефіцієнт технологічної підтримки комунікації, який враховує використання відповідних технологій для подолання комунікаційних бар'єрів.

У контексті розподілених проєктних команд географічна відповідність не обмежується лише фізичною відстанню між членами команди. Оцінка географічної відповідності ( $GSS_i$ ) повинна враховувати не лише кількість кілометрів між учасниками, але й рівень технологічного забезпечення для забезпечення ефективної комунікації. Застосування високотехнологічних рішень, як-от: відеоконференції, системи керування проєктами або інструменти співпраці в реальному часі, може значно зменшити “географічну відстань” навіть між командами, розташованими на різних континентах.

Отже,  $T_{ij}$  – це коефіцієнт, який враховує можливості подолання географічних відстаней за допомогою сучасних комунікаційних технологій. Наприклад, якщо команда має доступ до ефективних засобів комунікації, коефіцієнт  $T_{ij}$  буде високим, що збільшить загальний показник  $GSS_i$ , навіть якщо фактична фізична відстань між членами команди значна.

Перехід до фази прийняття рішень (Decision Conditions). Після розрахунків усіх показників за допомогою математичної моделі, система переходить до фази прийняття рішень (Decision Conditions – DC), де відбувається перевірка умов відбору кандидатів для формування оптимальної команди. На цій фазі здійснюється аналіз кожного кандидата за наступними умовами:

$$DC_i = \begin{cases} \text{Yes, якщо } FS_i > Threshold_{FS} \wedge TAS_i > Threshold_{TAS} \wedge GSS_i > Threshold_{GSS} \\ \text{No, в іншому випадку} \end{cases}, \quad (5)$$

- $DC_i$  – рішення щодо залучення кандидата  $i$  до команди. Значення “Yes” означає, що кандидат відповідає всім критеріям і може бути залучений до команди, натомість “No” означає, що кандидат не відповідає умовам і буде усунений.
- $FS_i$  (Functionality Score) – оцінка відповідності кандидата функціональним вимогам завданням. Цей показник оцінює знання, навички та досвід кандидата у контексті виконання конкретних завдань.
- $Threshold_{FS}$  – мінімальне значення оцінки функціональності (Functionality Score), яке кандидат повинен перевищити, щоб відповідати встановленим функціональним вимогам.
- $TAS_i$  (Temporal Alignment Score) – оцінка відповідності кандидата часовим параметрам, тобто його доступності для роботи у необхідні робочі години.
- $Threshold_{TAS}$  – мінімальне значення Temporal Alignment Score, яке кандидат повинен перевищити, щоб відповідати часовим параметрам.
- $GSS_i$  (Geographical Suitability Score) – оцінка відповідності кандидата географічним параметрам, яка враховує вплив його місцезнаходження на здатність ефективно взаємодіяти з іншими членами команди.
- $Threshold_{GSS}$  – мінімальне значення Geographical Suitability Score, яке кандидат повинен перевищити, щоб відповідати географічним параметрам.

Залежно від того, чи відповідає кандидат усім трьом умовам (функціональна, часова та географічна відповідність), його або залучають до команди або усувають. Цей процес реалізується ітеративно, доки не буде сформовано повний набір учасників проєктної команди.

Окрім фізичної відстані та технологічної підтримки, слід враховувати соціокомунікаційну віддаленість між учасниками команди. Ця характеристика передбачає культурні, ментальні, історичні та соціальні особливості, які можуть впливати на ефективність співпраці. Наприклад, відмінності у ментальності, релігії, звичаях, мові або соціальних нормах можуть створювати “відстані”, які не завжди можна подолати лише за допомогою технологій.

Соціокомунікаційна віддаленість є критично важливою в розподілених командах, оскільки вона впливає на динаміку співпраці навіть за фізичної близькості. Наприклад, робота у спільній проєктній команді представників різних національностей або релігій може бути ускладнена соціальними чи культурними бар’єрами, навіть якщо вони знаходяться в сусідніх містах.



Різні культури мають свої підходи до прийняття рішень, організації праці, спілкування і навіть розуміння часу. Одні культури схильні до ієрархічних структур і формальної комунікації, натомість інші віддають перевагу демократичному стилю управління. Такі відмінності можуть викликати непорозуміння та напруженість, якщо їх не врахувати. Мовні бар'єри, відмінності у способах мислення та соціальні норми також можуть впливати на взаємодію в команді.

Успішна робота розподілених команд залежить не лише від технологій, але й від того, як члени команди подолають культурні, мовні та ментальні бар'єри. Комплексний підхід до оцінки географічної відповідності допоможе підвищити ефективність комунікації та зменшити соціальні бар'єри між учасниками.

#### **Погодження та затвердження складу команди**

Після формування команди набір кандидатів передають на погодження стейкхолдерам та Product Owner. Погодження складу команди є важливим етапом, оскільки фіксує та підтверджує, що всі учасники команди відповідають вимогам проекту, і саме цей склад забезпечує ефективне розв'язання задач на основі пріоритизованих User Stories (US).

Отже, фаза Sprint Plan забезпечує оптимальний вибір команди на основі процедури оцінювання кандидатів, яка враховує функціональні, часові та географічні аспекти. Це своєю чергою гарантує, що завдання поточного спринту будуть виконані з максимальною ефективністю.

### **3. Беклог продукту (Product Backlog)**

Після етапу формування команди (Team Selection) система переходить до етапу беклогу продукту. Цей етап є ключовим для подальшого планування та організації робіт над проектом, оскільки саме тут відбувається детальний аналіз кожної історії користувача (User Story – US), а саме стислого опису кінцевого результату для конкретного користувача, що є запрошенням до обговорення деталей, на основі чого проводять оцінювання зусиль, необхідних для їхньої реалізації.

Оцінювання балами історії користувача (Story Point estimation):

*Analysis: for each User Story (US)  $1 \leq s \leq p$ , де* (6)

- Analysis означає процес детального аналізу кожної користувацької історії (User Story, US).
- for each User Story (US)  $1 \leq s \leq p$  означає, що аналізу піддаються всі історії користувачів, що були відібрані для спринту, де  $s$  представляє порядковий номер історії, а  $p$  – загальну кількість пріоритетних історій для поточного спринту.
- Story Point estimation — це процес оцінки складності або зусиль, необхідних для реалізації кожної історії. Story Points використовуються для оцінки та вимірювання розміру завдання в термінах складності, обсягу роботи або ризиків.

На цьому етапі відбувається оцінка кожної User Story (US) з точки зору складності та необхідних ресурсів для її реалізації. Основними вихідними залежностями цього етапу є Activity Dependency (AD), яка визначає, що аналіз кожної User Story має завершитися до того, як відбудеться перехід до наступної фази пріоритизації (Selection  $p \leq n$  US (prioritization)), і Sequential Dependency (SD), що відображає послідовність, згідно з якою всі попередні пріоритизовані User Stories повинні бути проаналізовані перед переходом до Sprint Backlog.

Аналіз кожної User Story є критичним етапом, який забезпечує основу для подальшого планування та реалізації. Після завершення цього аналізу система переходить до стану Sprint Backlog. У цьому стані системи відбувається створення завдань (Tasks) для кожної User Story на основі його аналізу.

### **4. Беклог спринта (Sprint Backlog)**

Упорядкований список роботи, що має бути виконана в спринті (беклог), є важливою складовою частиною Agile методології, яка використовується для організації та управління завданнями в межах спринту. В цьому стані системи фіксуються всі завдання, що повинні бути виконані

протягом поточного спринту, а також забезпечується безперервний моніторинг їхнього виконання. Sprint Backlog є динамічним документом, який постійно оновлюється на основі змінних умов та вимог проекту. Завдання, які містяться в Sprint Backlog, визначаються і формуються на основі аналізу User Stories (USs), проведеного на попередньому етапі, і проходять кілька стадій перевірки та затвердження.

На етапі **Analysis: for each User Story (US)  $1 \leq s \leq p$  (Story Point estimation)** було здійснено детальний аналіз кожного User Story, яка була відібрана для поточного спринту. Цей аналіз передбачає оцінювання складності процесів, необхідних ресурсів і часу, який знадобиться для виконання завдань, пов'язаних із кожною User Story. Всі ці оцінки консолідуються в **Sprint Backlog**, де створюються конкретні завдання (Tasks), що відповідають вимогам User Stories. **Sprint Backlog** залежить від даних, отриманих на попередніх етапах, і відображає логічну послідовність виконання завдань. У ньому враховуються всі аспекти, пов'язані з функціональністю, послідовністю, а також специфічні залежності між завданнями, які мають бути виконані.

FD (Functional Dependency) є ключовою залежністю, що відображає функціональні вимоги, які були закладені на етапі аналізу User Stories. Кожне завдання, залучене до Sprint Backlog, повинно чітко відповідати визначеним функціональним критеріям. Це забезпечує узгодженість завдань із загальними цілями проекту та гарантує, що кожне завдання сприяє досягненню конкретних функціональних результатів. Описана залежність фіксує той факт, що завдання в Sprint Backlog не лише розподіляються довільно, а мають конкретну функціональну мету, яка інтегрується в загальну архітектуру системи. Це допомагає уникнути дублювання зусиль, а також запобігає ситуаціям, коли певні функціональні елементи залишаються нерозробленими або неправильно реалізованими.

SD (Sequential Dependency) відображає послідовність виконання завдань у межах спринту. У рамках Sprint Backlog визначається порядок виконання завдань, що враховує їхній взаємозв'язок та залежність одне від одного. Наприклад, завдання, яке залежить від виконання іншого завдання, не може розпочатися, поки перше завдання не буде завершено. Ця залежність є важливою для підтримки логічної послідовності розроблення та тестування компонентів системи, що знижує ризики виникнення конфліктів під час інтеграції, покращує ефективність командної роботи та забезпечує своєчасне завершення спринту.

Sprint Backlog відіграє ключову роль у забезпеченні управління та контролю процесу розроблення протягом спринту. Він є не просто списком завдань, а інструментом, що забезпечує прозорість і видимість усього процесу розроблення для всієї команди. Усі учасники команди, зокрема розробники, тестувальники та Product Owner, мають доступ до Sprint Backlog, що дає їм змогу координувати свої дії, виявляти можливі проблеми на ранніх етапах та приймати оперативні рішення для їхнього усунення.

Окрім того, Sprint Backlog служить основою для оцінювання прогресу роботи команди. Протягом спринту завдання можуть змінювати свій статус (наприклад, з “в роботі” на “завершене”), що дає змогу команді постійно оцінювати свій прогрес і вчасно виявляти відхилення від запланованого графіка, а також виводити систематизовані дані на аналітичній панелі для Product Owner чи стейкхолдерів для оцінювання загального прогресу щодо проекту та визначати проблемні місця в роботі команди.

Після того, як усі завдання залучено до Sprint Backlog, відбувається перехід до стану:

**Проектування завдань (Designing task) set for US S:  $T_s \cong \{ T_i \mid 1 \leq i \leq sn, sn \in \mathbb{N} \}$** , де

- Designing task set for US S процес проектування набору завдань для конкретної користувачької історії (User Story), позначеної як S.
- $T_s$  – набір завдань, асоційованих з конкретною User Story S.
- $T_i$  – окреме завдання в межах набору завдань  $T_s$ .
- $1 \leq i \leq sn$  — індекс  $i$  приймає значення від 1 до  $sn$ , де  $sn$  — кількість завдань, які становлять набір для конкретної User Story.

- $sn \in \mathbb{N}$  — кількість завдань  $sn$  є натуральним числом.

На цьому етапі відбувається проектування завдань для кожного User Story, що забезпечує основу для їхньої реалізації. Завдання, сформовані на основі Sprint Backlog, проектуються з урахуванням усіх функціональних та послідовних залежностей, визначених раніше. Цей етап має зворотний зв'язок Activity Dependency (AD) з попередніми станами, що забезпечує логічну послідовність і узгодженість усього процесу розроблення. Додатково важливу роль на цьому етапі відіграє процес комунікації (CP), який забезпечує постійну комунікацію між Development Team та іншими стейкхолдерами, зокрема Product Owner. Ця комунікація є критичною для уточнення вимог, узгодження рішень і забезпечення того, що проектування завдань відповідає очікуванням замовника.

Беклог спринта є невід'ємною частиною управління проектом у межах Agile-методології. Відповідний підхід до управління "Sprint Backlog" забезпечує узгодженість між функціональними вимогами, логічною послідовністю виконання завдань та прозорістю процесу розроблення. Це своєю чергою підвищує ймовірність успішного завершення спринту, знижує ризики та забезпечує високу якість кінцевого продукту. Успішне впровадження Sprint Backlog вимагає не тільки правильної технічної реалізації, але й постійної уваги до комунікації між учасниками команди, а також регулярної оцінки прогресу та внесення необхідних коректив у процес розроблення.

### 5. Архітектура підсистеми (Sub-system Architecture)

На цьому етапі, після завершення формування Sprint Backlog, система переходить до стану Sub-system Architecture. Цей етап є критичним для забезпечення цілісності архітектури підсистеми та її подальшої інтеграції в загальну схему. Розглянемо детальніше кожен з елементів цього процесу, а також їхні взаємозв'язки та важливість для успішного управління розподіленими ІТ-проектами.

Завдання з програмування (Coding tasks):  $t \in T_s$ , де

- $t$  – окреме завдання, яке виконується під час розроблення програмного продукту;
- $t \in T_s$  – вказує, що завдання  $t$  є елементом набору завдань  $T_s$ , що були визначені для конкретної користувацької історії (User Story);
- $T_s$  – набір завдань, асоційованих з певною користувацькою історією (User Story S), які необхідно реалізувати під час розроблення.

Це етап, на якому відбувається реалізація конкретних завдань, що визначені у **Sprint Backlog** і спроектовані на попередньому етапі.

#### Проектування завдань (Designing task) set for US S: $T_s \cong \{ T_i \mid 1 \leq i \leq sn, sn \in \mathbb{N} \}$

Цей етап є ключовим для забезпечення коректної реалізації функціональності кожної User Story. Кодування є основою процесу розроблення програмного продукту, де всі попередні етапи (планування, аналіз, проектування) перетворюються на реальний програмний код.

Sequential Dependency (SD) – це вхідна залежність із етапу розроблення набору завдань (Designing task set for US S:  $T_s$ ). Ця залежність визначає, що процес кодування може розпочатися лише після завершення проектування відповідних завдань. Це гарантує, що всі архітектурні рішення, прийняті на етапі проектування, будуть враховані під час написання коду. Отже, забезпечується послідовність і узгодженість архітектурних вимог на кожному етапі.

Далі слідує Activity Dependency (AD) – вихідна залежність, яка встановлюється між кодуванням (Coding tasks) та наступними етапами, як-от тестування та інтеграція. Вона забезпечує послідовний перехід від одного етапу до іншого, гарантуючи плавний потік процесів і узгодженість результатів. Завдяки цій залежності досягається безперервність і зв'язність у процесі розроблення.

Поряд із цим важливим аспектом є ресурси (Resources). І на етапі розроблення набору завдань, і на етапі кодування є певна залежність від наявних ресурсів. Це означає, що для успішного виконання завдань необхідно забезпечити доступність усіх потрібних ресурсів, зокрема розробників, інструменти, доступ до баз даних, середовища розроблення тощо. Правильне планування і

використання ресурсів є критичним для забезпечення своєчасного та якісного завершення кодування. Отже, належне управління ресурсами відіграє важливу роль у процесах розроблення.

Останній аспект, який необхідно враховувати, – це взаємодія з командою розробників (Development Team). Етап кодування також суттєво залежить від команди розробників. Це підкреслює необхідність постійної взаємодії між розробниками, обміну інформацією та узгодження дій у процесі реалізації завдань. Враховуючи, що розроблення здійснюється розподіленими командами, важливо забезпечити ефективну комунікацію та координацію між членами команди, розташованими в різних часових та географічних зонах. Отже, забезпечується ефективність та узгодженість роботи команди на всіх етапах проекту.

## 6. Модулі (Modules)

Після завершення кодування відбувається перехід до стану Modules, де здійснюється агрегування модулів, які були реалізовані під час Coding tasks. Цей етап є критичним для перевірки цілісності та взаємодії всіх компонентів системи перед їхньою інтеграцією.

На етапі Modules здійснюється комплексне тестування реалізованих модулів. Це передбачає перевірку функціональності, стабільності, сумісності та інших важливих аспектів, які гарантують, що модулі відповідають вимогам і можуть бути інтегровані в систему без помилок.

Основні залежності для етапу тестування починаються з Activity Dependency (AD), що є вхідною залежністю від Coding tasks. Тож тестування не може розпочатися, поки всі завдання з кодування не будуть завершені. Ця залежність забезпечує послідовність і чітку організацію роботи, де кожен етап ґрунтується на результатах попереднього.

Після завершення кодування важливу роль відіграє Reciprocal Dependence Task (RDT). Ця залежність відображає взаємодію між тестуванням і процесом виправлення помилок. Якщо під час тестування виявляються помилки, вони передаються на етап кодування для виправлення, після чого знову проходять тестування. Цей цикл повторюється доти, поки всі помилки не будуть усунуті.

Одним із важливих елементів процесу тестування є Error Reports. Тестування генерує звіти про помилки, які потім використовуються для аналізу та виправлення проблем у коді. Ці звіти є критично важливими для виявлення недоліків і забезпечення їхнього усунення до переходу на наступний етап.

Далі слідує Sequential Dependency (SD), що є вихідною залежністю до наступного етапу. Тестування має бути завершене до того, як система перейде до стадії інтеграції. Це гарантує, що всі компоненти системи будуть інтегровані лише після того, як вони пройшли успішне тестування.

Нарешті, варто звернути увагу на ресурси (Resources). Подібно до кодування, тестування також залежить від наявності ресурсів, як-от: тестувальники, автоматизовані інструменти тестування, середовища для проведення тестів тощо. Належне планування ресурсів є ключовим фактором успішного завершення етапу тестування, що дає змогу уникнути затримок і забезпечити високу якість виконання завдань.

## 7. Безпомилкові модулі (Errorless Modules)

Після завершення тестування та виправлення всіх виявлених помилок система переходить до стану **Errorless Modules**. Це означає, що всі модулі були протестовані, і вони відповідають встановленим вимогам.

Після того, як всі модулі визнані безпомилковими, система переходить до інтеграції всіх компонентів у єдину функціональну систему. Інтеграція є складним процесом, який вимагає ретельної координації та перевірки на всіх рівнях, щоб забезпечити коректну системну роботу компонентів разом.

### Інтеграція усіх завдань (Integration of all tasks) $t \in T_s$ ,

- **інтеграція усіх завдань (Integration of all tasks)** – означає процес інтеграції всіх завдань, які були виконані та протестовані, у загальну систему;

- $t \in T_s$  – позначає, що кожне завдання  $t$  є частиною набору завдань  $T_s$ , який був раніше визначений для реалізації конкретної підсистеми або функції системи.

Ключовий етап у процесі розроблення програмного продукту, на якому окремі компоненти, що були розроблені та протестовані раніше, інтегруються в єдину функціональну систему. Цей процес є критичним, оскільки забезпечує цілісність і взаємодію всіх компонентів, що є необхідною передумовою для правильного функціонування кінцевого продукту. Основні залежності на цьому етапі гарантують, що інтеграція проходить успішно, а компоненти системи відповідають встановленим вимогам.

На етапі інтеграції важливу роль відіграє Activity Dependency (AD), яка вимагає, щоб усі модулі пройшли тестування і були визнані безпомилковими перед початком інтеграції. Цей фактор забезпечує те, що до системи залучаються лише коректно функціонуючі компоненти, а це гарантує стабільність та надійність інтегрованої системи.

Наступний аспект – це Fit Dependency (FD) – забезпечує те, що всі модулі, які інтегруються, відповідають один одному та архітектурним вимогам системи. Водночас підтверджується, що компоненти працюють разом без конфліктів, створюючи гармонійну і функціонально узгоджену систему.

Sequential Dependency (SD) визначає послідовність етапів інтеграції, які повинні бути дотримані для забезпечення правильного функціонування системи. Всі модулі повинні бути інтегровані в певному порядку, щоб уникнути можливих збоїв у роботі системи. Це створює чіткий план дій і забезпечує логічний та структурований процес інтеграції.

Нарешті, Communication Process (CP) забезпечує постійну комунікацію між командою розроблення та іншими зацікавленими сторонами під час інтеграції. Це допомагає оперативно вирішувати проблеми, що виникають, і забезпечує відповідність кінцевого продукту вимогам замовника. Така комунікація є критично важливою для успішної інтеграції, оскільки дає змогу швидко реагувати на зміни та координувати зусилля всіх учасників процесу.

## 8. Функціональна підсистема (Functional Subsystem)

Процес розроблення програмного забезпечення, особливо в контексті розподілених проектних команд, вимагає особливої уваги до координації, управління залежностями та контролю якості на всіх етапах. Розглянемо детальніше фази “Functional Subsystem” та “User Acceptance”, зокрема їхню взаємодію, значення для успішного завершення проєкту, і як саме вони впливають на роботу розподіленої проєктної команди.

Після успішної інтеграції всіх компонентів на етапі інтеграції, система переходить до фази Functional Subsystem. На цьому етапі всі компоненти об’єднуються в підсистему, яка повинна відповідати визначеним функціональним вимогам проєкту. Functional Subsystem є критичною фазою, оскільки в процесі їхньої реалізації визначається, наскільки ефективно модулі системно взаємодіють для забезпечення необхідної функціональності системи.

Стан **Sub-system Delivery** передбачає доставку підсистеми замовникові або кінцевому користувачу для подальшого тестування та оцінки. Цей стан є завершальним на цьому етапі розроблення, де всі компоненти повинні бути готові для передачі та повністю відповідати встановленим вимогам. Це означає, що підсистема повинна бути протестована і визнана функціональною та безпомилковою.

Робота розподілених команд на етапі Functional Subsystem може бути організована за трьома основними типами розподілу: функціональним, географічним та часовим. Функціональний розподіл передбачає, що кожна команда або підгрупа команди відповідає за реалізацію певної функціональності системи. Така організація забезпечує спеціалізацію й підвищує ефективність роботи, оскільки команди можуть зосередитися на конкретних аспектах проєкту.

Географічний розподіл передбачає, що команди можуть бути розміщені в різних локаціях або навіть країнах. Це ускладнює процес інтеграції через необхідність врахування різниць у часових

поясах та культурних особливостях. Однак цей підхід дає змогу залучити експертів із різних частин світу, що може значно покращити якість кінцевого продукту.

Часовий розподіл своєю чергою передбачає, що команди працюють у різних часових поясах, а це забезпечує безперервний процес розроблення. У такій організації одна команда може продовжувати роботу після завершення робочого дня іншої команди, що значно прискорює процес розроблення. Проте цей підхід вимагає ретельного планування та координації, щоб уникнути помилок і непорозумінь у синхронізації робочих процесів.

### 9. Прийняття користувачем (User Acceptance)

Фаза User Acceptance є однією з найважливіших у процесі розроблення, оскільки саме тут здійснюється оцінка того, наскільки готова система до використання кінцевим користувачем. Усі функції, інтегровані в систему, перевіряються на відповідність вимогам замовника і кінцевого користувача. Йдеться про різні види тестування, зокрема функціональні тести, юзабіліті-тести, навантажувальні тести тощо.

Згаданий етап User Acceptance системи передбачає проходження через стан “System complete”, який виконує роль ключової точки прийняття рішення. На цьому етапі здійснюється оцінка готовності системи до релізу. Якщо система відповідає всім встановленим вимогам, пройшла необхідні тести і готова до випуску або передачі кінцевому користувачу, відбувається перехід до стану завершення проєкту (End). У цьому випадку система визнається готовою до використання. Однак якщо система не відповідає вимогам або під час тестування були виявлені критичні помилки, ухвалюється рішення про повернення до попередніх етапів проєкту, як-от фаза Identification: n USs та Sprint plan. Всі виявлені недоліки передаються на доопрацювання, а система підлягає вдосконаленню до моменту повторної оцінки користувачами.

Для розподілених команд важливо забезпечити узгодженість та своєчасність тестування на етапі User Acceptance. Кожен член команди повинен чітко розуміти свої обов'язки та межі відповідальності, щоб уникнути дублювання зусиль або пропуску важливих тестів. Координація між членами команди, особливо в умовах географічного та часового розподілу, має бути налагоджена так, щоб забезпечити своєчасне виявлення та усунення проблем.

Розроблена діаграма (рис. 1) відображає цілісний системний підхід до управління розподіленими проектними командами в умовах функціональної, часової та географічної розподіленості. Запропонована концепція взаємодії дає змогу оптимізувати процеси розроблення програмного забезпечення, в контексті реалізації високотехнологічних складних сучасних ІТ-проєктів, де ефективно управління розподіленими командами стає критичним фактором успіху.

Діаграма (рис. 1) чітко ілюструє, як саме всі етапи розроблення пов'язані між собою через залежності та послідовності. Це забезпечує логічний та узгоджений перехід від одного етапу до іншого, що є необхідним для підтримання високого рівня контролю за якістю та ефективністю роботи. Особливу увагу приділено питанням управління розподіленими командами, що працюють в різних часових поясах та географічних локаціях. Діаграма, яка за своєю суттю є функціональною моделлю, враховує функціональний розподіл завдань між членами команди, що дає змогу ефективно використовувати час та ресурси, забезпечуючи безперервний процес розроблення.

Запропонований підхід включає всі етапи та процеси забезпечення якості кінцевого продукту, зокрема модульне тестування, інтеграцію компонентів, перевірку функціональності системи на етапі User Acceptance та фінальний контроль якості перед випуском продукту. Це значно знижує ймовірність помилок та забезпечує високу якість кінцевого продукту. Запропонована модель є достатньо гнучкою для адаптації до різних типів проєктів та специфічних вимог замовника. Вона може слугувати універсальним інструментом, який може бути використаний для різних типів ІТ-проєктів незалежно від їхньої складності та масштабу.

Одним із напрямків подальших досліджень буде оптимізація процесів комунікації між членами розподілених команд. Це може передбачати розробку нових чи інтеграцію наявних інстру-

ментів та методів для забезпечення безперебійного та ефективного обміну інформацією між командами, що працюють у різних часових поясах та географічних регіонах. Подальші дослідження будуть спрямовані на адаптацію та масштабування моделі для управління великими проєктами з великою кількістю учасників та високою складністю. Загалом йдеться про формування системних цілісних процесів та процедур функціонування сучасних ІТ-компаній та бізнесів, які щораз більшою мірою трансформуються у великі розподілені проєктні колективи та команди. Це передбачає інтеграцію додаткових механізмів контролю за ресурсами, управління знаннями та забезпечення якості.

### Висновки

У статті проаналізовано проблеми координації та розподілу завдань у розподілених Agile-командах, а також запропоновано новий функціональний підхід до їхнього вирішення. Основну увагу зосереджено на моделі, яка враховує різноманітні фактори, як-от досвід членів команди, їхня продуктивність та специфічні вимоги до ролей у проєкті. Запропонований підхід спрямований на підвищення ефективності управління проєктами в умовах розподілених команд, зокрема, шляхом забезпечення прозорого та обґрунтованого розподілу завдань в чітко окресленій послідовності виконання завдань проєкту.

У роботі запропоновано функціональну модель, що подана у вигляді чіткої послідовності певних етапів, які реалізуються в межах автоматизованої оцінки та формування розподіленої команди на основі трьох основних факторів: функціональність, часова доступність та географічна відповідність. Це своєю чергою забезпечує якісний підбір членів команди, мінімізує ризики та підвищує ефективність виконання завдань, що є ключовим для успішного втілення проєкту.

Подана модель є інструментом, який забезпечує комплексний системний підхід до управління розподіленими ІТ-проєктами. Вона враховує низку важливих аспектів, що необхідні для успішного завершення проєкту, і може бути успішно використана для підвищення якості та ефективності роботи розподілених проєктних команд. Запропонована функціональна модель є гнучкою, адаптивною і має значний потенціал для подальшого розвитку та вдосконалення. Проведена авторами робота не тільки забезпечує вдосконалення наявних процесів, але й відкриває нові можливості для дослідження і запровадження інновацій у сфері управління розподіленими проєктними командами, зокрема в галузі ІТ.

Подальші перспективи дослідження будуть зацентровані на вдосконаленні розробленої функціональної моделі для її адаптації до більш складних проєктів із великою кількістю учасників і високою динамікою змін. Особливу увагу буде приділено інтеграції нових інформаційних інструментів для забезпечення ефективної комунікації та координації між членами розподілених команд, а також розвитку підходів до подолання культурних, часових та соціальних бар'єрів у масштабних ІТ-орієнтованих проєктах.

### СПИСОК ЛІТЕРАТУРИ

1. Project Management Institute. (n.d.). *Pulse of the profession: Future of project work*. PMI. <https://www.pmi.org/learning/thought-leadership/pulse/future-of-project-work>
2. Atlassian. (n.d.). *Scrum artifacts*. Atlassian. <https://www.atlassian.com/agile/scrum/artifacts>
3. Васьків, Р., Веретеннікова, Н. (2024). Інформаційні та комунікаційні інструменти ефективного функціонування розподілених проєктних команд. *Вісник Національного університету "Львівська політехніка" "Інформаційні системи та мережі"*, 15, 357–369. <https://doi.org/10.23939/sisn2024.15.357>
4. Van Steen, M., Tanenbaum, A. S. (2023). *Distributed systems* (4th ed.). distributed-systems.net. <https://www.distributed-systems.net/index.php/books/ds4/>
5. Forsgren, N., Humble, J., Kim, G. (2018). *Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations*. IT Revolution Press. <https://itrevolution.com/book/accelerate/>

6. Lamersdorf, A., Münch, J., Rombach, D., Sihling, M., Natschläger, C. (2012). A rule-based model for customized risk identification and evaluation of task assignment alternatives in distributed software development projects. *Journal of Software: Evolution and Process*, 24(7), 713–727. <https://doi.org/10.1002/smr.559>
7. Mahmood, S., Anwer, S., Niazi, M., Alshayeb, M., & Richardson, I. (2017). Key factors that influence task allocation in global software development. *Information and Software Technology*, 91, 102–122. <https://doi.org/10.1016/j.infsof.2017.06.009>
8. Simão Filho, M., Pinheiro, P. R., Albuquerque, A. B., Barreto, A. (2019). Task allocation and coordination in distributed agile software development: A systematic review. *Complexity*, 1–22. <https://doi.org/10.1155/2019/7015418>
9. Pereira, L., Jerónimo, C., Simões, F., Sousa, P. (2024). Project virtual teams: Systematic literature review. *International Journal of Agile Systems and Management*, 17(1), 15–45.
10. Aslam, W., Ijaz, F. (2018). A quantitative framework for task allocation in distributed agile software development. *IEEE Access*, 6, 15380–15390. <https://doi.org/10.1109/ACCESS.2018.2812523>
11. Stray, V., Moe, N. B. (2020). Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software*, 170, 110717. <https://doi.org/10.1016/j.jss.2020.110717>
12. Patel, R., Rudnick-Cohen, E., Azarm, S., Otte, M., Xu, H., Herrmann, J. W. (2020). Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. <https://doi.org/10.1109/ICRA40945.2020.9197314>
13. Nundlall, C., Nagowah, S. D. (2021). Task allocation and coordination in distributed agile software development: A systematic review. *Journal of Software Engineering Research and Development*, 13, 321–330. <https://doi.org/10.1007/s40411-020-00130-0>
14. Bick, S., Spohrer, K., Hoda, R., Scheerer, A., Heinzl, A. (2018). Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering*, 44(10), 932–950. <https://doi.org/10.1109/TSE.2017.2730870>

## REFERENCES

1. Project Management Institute. (n.d.). *Pulse of the profession: Future of project work*. PMI. <https://www.pmi.org/learning/thought-leadership/pulse/future-of-project-work>
2. Atlassian. (n.d.). *Scrum artifacts*. Atlassian. <https://www.atlassian.com/agile/scrum/artifacts>
3. Vaskiv, R., Veretennikova, N. (2024). *Information and Communication Tools for Effective Functioning of Distributed Project Teams*. *Journal of Lviv Polytechnic National University “Information Systems and Networks”*, 15, 357–369. <https://doi.org/10.23939/sisn2024.15.357>
4. Van Steen, M., Tanenbaum, A. S. (2023). *Distributed systems* (4th ed.). distributed-systems.net. <https://www.distributed-systems.net/index.php/books/ds4/>
5. Forsgren, N., Humble, J., Kim, G. (2018). *Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations*. IT Revolution Press. <https://itrevolution.com/book/accelerate/>
6. Lamersdorf, A., Münch, J., Rombach, D., Sihling, M., Natschläger, C. (2012). A rule-based model for customized risk identification and evaluation of task assignment alternatives in distributed software development projects. *Journal of Software: Evolution and Process*, 24(7), 713–727. <https://doi.org/10.1002/smr.559>
7. Mahmood, S., Anwer, S., Niazi, M., Alshayeb, M., Richardson, I. (2017). Key factors that influence task allocation in global software development. *Information and Software Technology*, 91, 102–122. <https://doi.org/10.1016/j.infsof.2017.06.009>
8. Simão Filho, M., Pinheiro, P. R., Albuquerque, A. B., Barreto, A. (2019). Task allocation and coordination in distributed agile software development: A systematic review. *Complexity*, 1–22. <https://doi.org/10.1155/2019/7015418>
9. Pereira, L., Jerónimo, C., Simões, F., Sousa, P. (2024). Project virtual teams: Systematic literature review. *International Journal of Agile Systems and Management*, 17(1), 15–45.
10. Aslam, W., Ijaz, F. (2018). A quantitative framework for task allocation in distributed agile software development. *IEEE Access*, 6, 15380–15390. <https://doi.org/10.1109/ACCESS.2018.2812523>



11. Stray, V., Moe, N. B. (2020). Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software*, 170, 110717. <https://doi.org/10.1016/j.jss.2020.110717>
12. Patel, R., Rudnick-Cohen, E., Azarm, S., Otte, M., Xu, H., Herrmann, J. W. (2020). Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. <https://doi.org/10.1109/ICRA40945.2020.9197314>
13. Nundlall, C., Nagowah, S. D. (2021). Task allocation and coordination in distributed agile software development: A systematic review. *Journal of Software Engineering Research and Development*, 13, 321–330. <https://doi.org/10.1007/s40411-020-00130-0>
14. Bick, S., Spohrer, K., Hoda, R., Scheerer, A., Heinzl, A. (2018). Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering*, 44(10), 932–950. <https://doi.org/10.1109/TSE.2017.2730870>

## PROCESS MANAGEMENT IN DISTRIBUTED PROJECT TEAMS

Roman Vaskiv<sup>1</sup>, Nataliia Veretennikova<sup>2</sup>

<sup>1,2</sup>Lviv Polytechnic National University,

Information Systems and Networks Department, Lviv, Ukraine

<sup>1</sup>E-mail: roman.i.vaskiv@lpnu.ua, ORCID: 0000-0002-8549-5035

<sup>2</sup>E-mail: nataliia.v.veretennikova@lpnu.ua, ORCID: 0000-0001-9564-4084

© Vaskiv R., Veretennikova N., 2024

**The paper examines approaches to managing the processes of coordination and distribution of tasks in distributed project teams in the IT industry, which work in geographically dispersed Agile environments. The focus is on developing a team member selection model that considers the experience, performance, and geographic location of members of distributed teams for effective task performance. The proposed functional model makes it possible to take into account the main factors and dependencies affecting the decision-making process in distributed teams, in particular functionality, time availability and geographical compatibility. This helps to minimize risks and increase the efficiency of project management, which is especially important in the rapidly changing IT market. The limitations of the model are analyzed and ways of further exploration are suggested.**

**Keywords: distributed project teams, Agile, distributed software development, project process management.**