ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Імітаційна модель роботи верстатного процесу систем масового обслуговування»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Семі АЛ ХАДАМ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Семі АЛ ХАДАМ
Ім'я, ПРІЗВИЩЕ
Керівник: Ігор БУЧЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент:
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
_____
«_____» _____ 2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ал Хадам Семі Резгович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Імітаційна модель роботи верстатного процесу систем масового обслуговування
керівник кваліфікаційної роботи Ігор БУЧЕНКО,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56
2. Строк подання кваліфікаційної роботи «5» червня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література; система масового обслуговування; дискретна математика; алгоритм імітації моделі; мова програмування: Java;
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Концептуальна модель системи
2. Формалізована модель системи
3. Програмна реалізація імітаційної моделі системи
4. Експериментальне дослідження моделі
5. Інтерпретація результатів моделювання
6. Висновки
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2025	Виконано
2	Опис концептуальної моделі	02.03-09.04.2025	Виконано
3	Створення формалізованої моделі	09.04-12.04.2025	Виконано
4	Реалізація моделі	12.04-16.04.2025	Виконано
5	Проведення експериментів	13.04-17.05.2025	Виконано
6	Висновки за результатами аналізу	17.05-20.05.2025	Виконано
7	Розробка презентації	20.05-25.05.2025	Виконано
8	Передзахист	27.05-29.05.2025	Виконано
9	Здача в деканат	02.06-05.06.2025	Виконано

Здобувач вищої освіти _____ Семі АЛ ХАДАМ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ігор БУЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 49 стор., 24 рис., 3 табл., 30 джерел.

Мета роботи – дослідження функціонування верстата у виробничому процесі, оцінка його завантаження та часу виконання завдань з урахуванням випадкових факторів, таких як поломки, час налаштування та змін у черзі завдань.

Об'єкт дослідження – процес обробки замовлення клієнта в автоматизованій інформаційній системі

Предмет дослідження – імітаційна модель роботи верстатного процесу систем масового обслуговування

Короткий зміст роботи: верстатні процеси у виробничих цехах часто стикаються з випадковими перервами через поломки та необхідністю переналагодження, що призводить до простоїв і зниження продуктивності. Щоб мінімізувати ці втрати, важливо заздалегідь оцінити поведінку верстата за різних умов експлуатації та експериментальних налаштувань. У роботі розглядається імітаційна модель верстатного процесу на основі формалізованого опису мережі масового обслуговування з урахуванням експоненційного розподілу інтервалів між надходженнями завдань, нормального розподілу часу обробки та ерлангового розподілу тривалості ремонту. Модель доповнена механізмом перенаправлення надмірних завдань до субпідрядника під час ремонту та можливістю переривання ремонту для пошуку запасних частин. Проведено верифікацію коректності алгоритму, а серія експериментів із варіюванням параметрів показала, що додаткові умови дозволяють суттєво знизити середній час перебування завдання в системі та оптимізувати завантаження верстата.

Результати дослідження підтверджують практичну цінність моделі для прийняття управлінських рішень щодо організації ремонтних і виробничих процесів.

КЛЮЧОВІ СЛОВА: ІМІТАЦІЙНА МОДЕЛЬ, МЕРЕЖА МАСОВОГО ОБСЛУГОВУВАННЯ, ЕКСПОНЕНЦІЙНИЙ РОЗПОДІЛ, НОРМАЛЬНИЙ РОЗПОДІЛ, ЕРЛАНГОВИЙ РОЗПОДІЛ, ДИСКРЕТНО-ПОДІЙНЕ МОДЕЛЮВАННЯ

ЗМІСТ

ВСТУП……...8
1 ПОНЯТТЯ ІМІТАЦІЙНА МОДЕЛЬ. СИСТЕМА МАСОВОГО ОБСЛУГОВУВАННЯ……..9
1.1 Імітаційна модель та її роль у дослідженнях……9
1.2 Система масового обслуговування……...…10
2 КОНЦЕПТУАЛЬНА ТА ФОРМАЛІЗОВАНА МОДЕЛЬ СИСТЕМИ……11
2.1 Звичайний варіант концептуальної системи……...…11
2.2 Модифікований варіант концептуальної системи……14
2.3 Звичайний варіант формалізованої системи……..17
2.4 Звичайний варіант формалізованої системи……..22
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ……...…26
3.1 Аргументація обраних засобів реалізації……26
3.2 Опис програмної реалізації імітаційної моделі……..28
3.3 Опис програмної реалізації модифікованої імітаційної моделі……37
3.4 Аргументація обраних засобів реалізації……41
4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ……...……...47
4.1 Дослідження моделі……..47
ВИСНОВКИ……52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……...53
Додаток А……56
Додаток Б……74
Додаток В……75
Додаток Г……78
Додаток Д……83

ВСТУП

Виробництво висуває високі вимоги до ефективності та надійності роботи обладнання, що є важливим чинником у забезпеченні безперервності технологічного процесу та підвищенні продуктивності підприємства. Одним із ключових елементів виробничого процесу є верстат, робота якого залежить від таких факторів, як розподіл завдань, налаштування, поломки та їхній ремонт.

У даному дослідженні здійснюється моделювання роботи верстата, який обробляє завдання в умовах випадкових перерв через поломки та враховує можливість перенаправлення завдань до субпідрядника при перевантаженні черги.

Метою роботи є дослідження функціонування верстата у виробничому процесі, оцінка його завантаження та часу виконання завдань з урахуванням випадкових факторів, таких як поломки, час налаштування та змін у черзі завдань.

Для досягнення поставленої мети буде розроблено імітаційну модель роботи верстата та її модифіковану версію на основі принципів теорії масового обслуговування та дискретно-подійного моделювання.

Це дослідження спрямоване на поліпшення організації роботи верстата шляхом аналізу ефективності виробничого процесу за різних сценаріїв. Результати роботи можуть бути взяті за основу для прийняття оптимальних управлінських рішень щодо використання обладнання, зменшення часу простою та оптимізації завантаження системи.

1 ПОНЯТТЯ ІМІТАЦІЙНА МОДЕЛЬ. СИСТЕМА МАСОВОГО ОБСЛУГОВУВАННЯ

1.1 Імітаційна модель та її роль у дослідженнях

Імітаційна модель — це абстрактне подання реального процесу або системи, яке дозволяє відтворювати й аналізувати її поведінку без необхідності фізичного втручання. Імітаційне моделювання широко застосовується в різних галузях, таких як виробництво, транспорт, логістика, банківська справа, медицина та інші. Воно дозволяє оцінювати ефективність систем, аналізувати різні сценарії розвитку подій, прогнозувати можливі проблеми та знаходити оптимальні рішення.

Важливу роль імітаційні моделі відіграють в умовах, коли дослідження фізичних систем або процесів може бути дорогою або технічно складною задачею. За допомогою моделювання можна проводити серію експериментів, що дає змогу зрозуміти, як зміни в параметрах системи впливають на її роботу, без реального втручання в процеси.

Імітаційні моделі використовуються для:

- оцінки ефективності виробничих процесів;
- прогнозування результатів роботи системи за різних умов;
- оптимізації параметрів системи для покращення її роботи;
- оцінки ризиків та ідентифікації потенційних проблем.

У контексті виробництва, імітаційне моделювання дає змогу оцінити, як змінюються показники системи за різних варіантів управлінських рішень, таких як зміна кількості робочих, налаштувань верстатів, планування ремонтних робіт та інше.

1.2 Система масового обслуговування

Система масового обслуговування (СМО) — це математична модель, що описує процеси обслуговування в умовах великої кількості клієнтів, які надходять до обслуговуючих пристроїв (як правило, сервісних, виробничих чи транспортних систем). В основі СМО лежить теорія черг, яка дозволяє математично описати як клієнти надходять до системи, як вони обслуговуються, і як система реагує на зміну параметрів навантаження.

Система масового обслуговування характеризується такими основними параметрами:

- *інтенсивність надходження клієнтів (λ)* — Це швидкість надходження клієнтів до системи;
- *час обслуговування клієнта (μ)* — Середній час, необхідний для обслуговування одного клієнта;
- *кількість серверів (s)* — Кількість обслуговуючих пристроїв, які працюють паралельно;
- *розподіл часу обслуговування* — Який може бути експоненціальним, нормальним або іншим;
- *черга* — Місце, де клієнти чекають, якщо всі сервери зайняті.

Типовим прикладом системи масового обслуговування є процес обслуговування клієнтів у банку, де клієнти чекають в черзі для того, щоб здійснити певну операцію. Враховуючи різні параметри (наприклад, кількість касирів та швидкість обслуговування), можна оцінити середній час очікування клієнтів, ймовірність того, що черга буде порожньою, або ймовірність того, що клієнти не зможуть бути обслуговані через перевантаження системи.

- завдяки аналізу та моделюванню таких систем можна;
- підвищити ефективність обслуговування;
- знизити час очікування клієнтів;
- прогнозувати можливі перевантаження в системі і планувати резерви.

2 КОНЦЕПТУАЛЬНА ТА ФОРМАЛІЗОВАНА МОДЕЛЬ СИСТЕМИ

2.1 Звичайний варіант концептуальної системи

Система, що моделюється, включає один верстат, який виконує обробку завдань у виробничому процесі. Завдання надходять до черги, виконуються на верстаті та після цього залишають систему. Робота верстата характеризується періодичними поломками, що впливають на час виконання завдань.

Система включає наступні елементи: черга завдань — місце накопичення завдань, що очікують обробки, верстат — основний елемент системи, що виконує завдання, система ремонту — підсистема, яка відповідає за усунення поломок верстата.

Вхідні змінні моделі визначають випадкові процеси, які впливають на функціонування системи(табл.2.1).

Таблиця 2.1 — Вхідні змінні моделі

	Початкове значення (год)	Діапазон варіювання (год)	Закон розподілу	Математичне сподівання
Час між надходженнями завдань	1	[0; ∞)	експоненціальний	$\lambda = 1$
Час налаштування верстата	0,35	[0,2; 0,5]	рівномірний	-
Час виконання завдання	0,5	[0,3; 0,7]	нормальний	$\mu = 0,5$
Час до поломки верстата	20	[15; 25]	нормальний	$\mu = 20$
Загальний час ремонту верстата	2,25	[1,5; 3]	ерланговий	$\lambda = 3/4$

Параметри моделі:

– кількість пристроїв кожної СМО – 1;

– кількість замовлень, які циркулюють всередині фрагменту поломки – 1;

– кількість етапів технологічної обробки деталі дорівнює 3;

– створення задачі для верстату, наладка верстату, виконання завдання;

– кількість фаз ремонту верстата. Три фази ремонту, що є характеристикою структури процесу усунення поломки, а також створення задачі на усунення поламки, та процес виникнення поломки;

– черги мережі мають необмежений розмір.

Вихідні змінні є результатами моделювання:

– завантаження верстата, частка часу, протягом якого верстат виконує обробку. Числові значення: від 0 до 1 (в частках часу);

– час виконання завдання. Час від надходження завдання до його виходу із системи. Статистичні показники включають: мінімальне максимальне та середнє значення змінної, а також квадратичне відхилення. Одиниця виміру години.

Система функціонує за наступною схемою(рис.2.1). Завдання надходять до черги перед верстатом. У разі готовності верстата та наявності завдань у черзі виконується їх обробка. Черга реалізується як FIFO (англ. first in, first out) — перший прийшов перший вийшов). Перед початком виконання кожного завдання відбувається налаштування верстата із випадковим часом. Виконання завдання може бути перервано через поломку, у такому випадку воно повертається на початок черги, а робота верстата зупиняється. При поломці верстата запускається процес ремонту, що складається з трьох послідовних незалежних фаз із випадковою тривалістю. Завдання, виконані на верстаті, залишають систему.

У звичайній моделі визначаються такі цільові показники, як завантаження верстата та статистичні показники часу виконання завдання. Завантаження верстата розраховується як співвідношення часу, протягом якого верстат виконує завдання та перебуває в стані наладки, до загального часу моделювання. Час виконання завдання включає тривалість його наладки, виконання, а також можливі затримки через чергу та поломки верстата, до поки завдання не буде виконано і не

покине систему. Для визначення даного показника буде обраховано середній, мінімальний та максимальний час виконання завдання системою, а також середнє відхилення. В результаті моделювання будемо мати достатньо велику кількості оброблених деталей, тож завданнями, що на момент завершення імітації не завершили свою обробку знехтуємо.

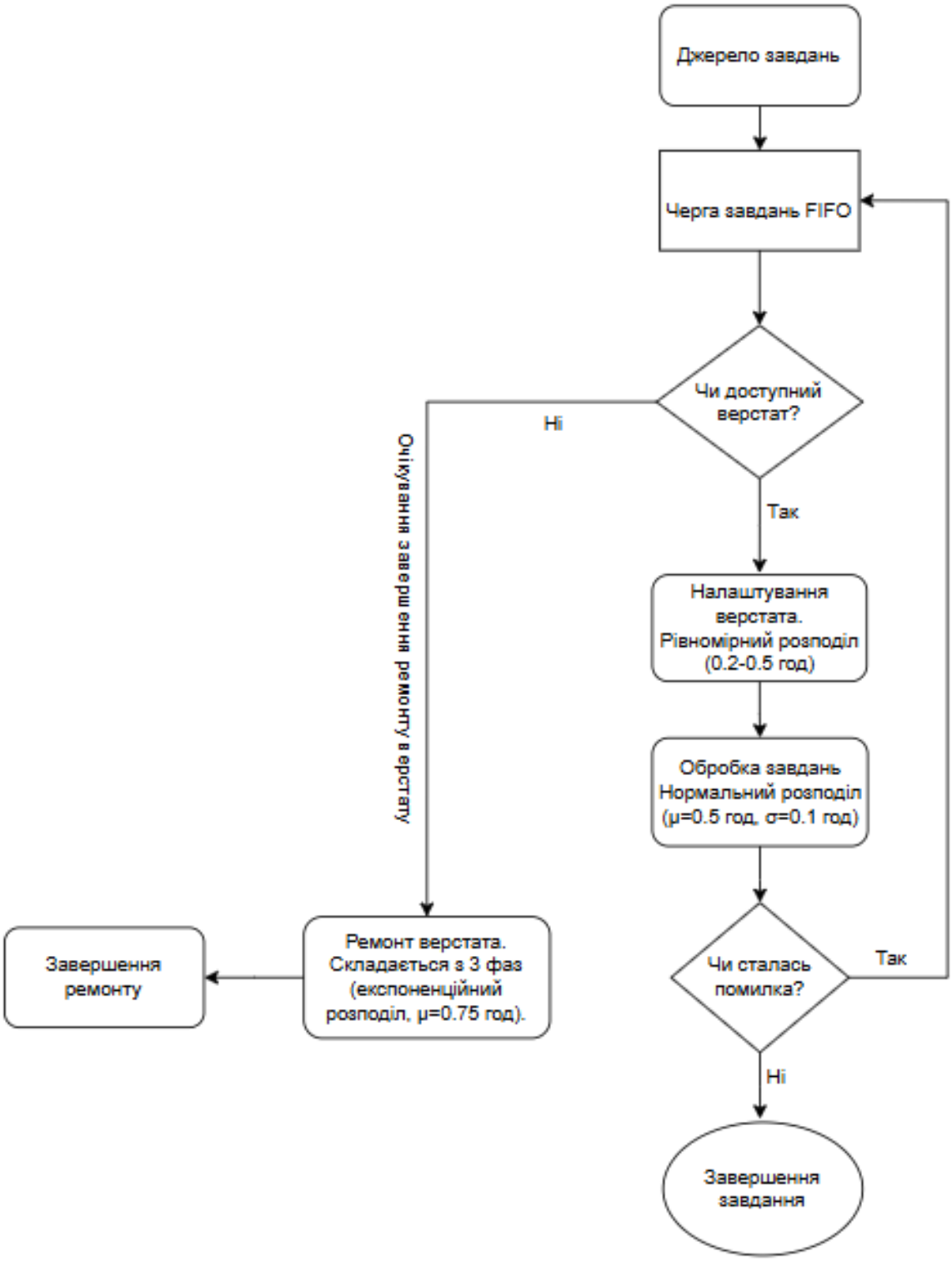


Рисунок 2.1 – Схематично зображене функціонування системи

2.2 Модифікований варіант концептуальної системи

Модифікована модель враховує перенаправлення завдань до субпідрядника у разі перевантаження черги під час ремонту та переривання ремонту для отримання запасних частин.

Система включає наступні підсистеми: черга завдань – місце накопичення завдань перед верстатом, верстат – основний елемент системи, що виконує обробку завдань, система ремонту – підсистема, яка займається усуненням поломок та субпідрядник – зовнішній елемент, що приймає надлишок завдань із черги.

Таблиця 2.2 — Вхідні змінні модифікованої моделі

	Початкове значення (год)	Діапазон варіювання (год)	Закон розподілу	математичне сподівання
Час між надходженнями завдань	1	[0; ∞)	експоненціальний	λ = 1
Час налаштування верстата	0,35	[0,2; 0,5]	рівномірний	-
Час виконання завдання	0,5	[0,3; 0,7]	нормальний	μ = 0,5
Час до поломки верстата	20	[15; 25]	нормальний	μ = 20
Загальний час ремонту верстата	2,25	[1,5; 3]	ерланговий	λ = 3/4
Час до переривання ремонту для отримання запасних частин	100	[50; 150]	експоненціальний	λ=100

Параметри моделі:

- кількість пристроїв кожної СМО – 1;
- кількість замовлень, які циркулюють всередині фрагменту поломки – 1;

– кількість етапів технологічної обробки деталі дорівнює 3;

– створення задачі для верстату, наладка верстату, виконання завдання;

– кількість фаз ремонту верстата. Три фази ремонту, що є характеристикою структури процесу усунення поломки, а також створення задачі на усунення поломки та процес виникнення поломки;

– черги лишаються необмеженого розміру, але встановлено фіксоване обмеження при виникненні поломки верстату: не більше трьох завдань у черзі перед верстатом, тобто за умови, що верстат зламаний, із черги будуть вилучатись завдання так, щоб сума завдань у черзі не перевищувала три, та переводитись до субпідрядника.

Вихідні змінні. Модифікація системи додає новий шлях для завдань — вони можуть при перенавантажені направлятися до субпідрядника в момент ремонту верстату. Частка таких завдань є новою змінною, яку необхідно врахувати. Переривання ремонту, вплине на середній час простою верстата та середній час перебування завдання в системі, також врахуємо частку часу переривань ремонту до загального часу ремонту.

– Завантаження верстата. Частка часу, протягом якого верстат безпосередньо виконує завдання та налаштовується. Важливо для порівняння з базовою моделлю;

– час виконання завдання. Час від надходження завдання до його виходу із системи. Статистичні показники включають: мінімальне максимальне та середнє значення змінної, а також квадратичне відхилення. Одиниця виміру години;

– частка завдань, переданих на субпідрядника. Кількість завдань, переправлених на субпідрядника, через перевантаження системи під час поломки верстата, розділена на загальну кількість завдань, що надійшли у систему та були нею оброблені. Дозволяє оцінити ступінь використання субпідрядника. Додано для більш наочного розуміння впливу модифікації на модель;

– частка часу витраченого на пошук запасних деталей, до загального часу ремонту верстата. Дозволяє оцінити вплив переривань на ремонт і відповідно, на

завантаження верстату. Додано для більш наочного розуміння впливу модифікації на модель.

Завдання надходять до черги перед верстатом. У разі готовності верстата та наявності завдань у черзі виконується їх обробка. Черга реалізується як FIFO. Перед початком виконання кожного завдання відбувається налаштування верстата із випадковим часом. Виконання завдання може бути перервано через поломку, у такому випадку воно повертається на початок черги, а робота верстата зупиняється. Якщо черга перевищує три завдання під час поломки, надлишок завдань, а також незавершене завдання передаються до субпідрядника. При поломці верстата запускається процес ремонту, що складається з трьох послідовних незалежних фаз з випадковою тривалістю. Ремонт може бути перервано випадковим чином на 3 години для отримання запасних частин. Завдання, виконані на верстаті, залишають систему.

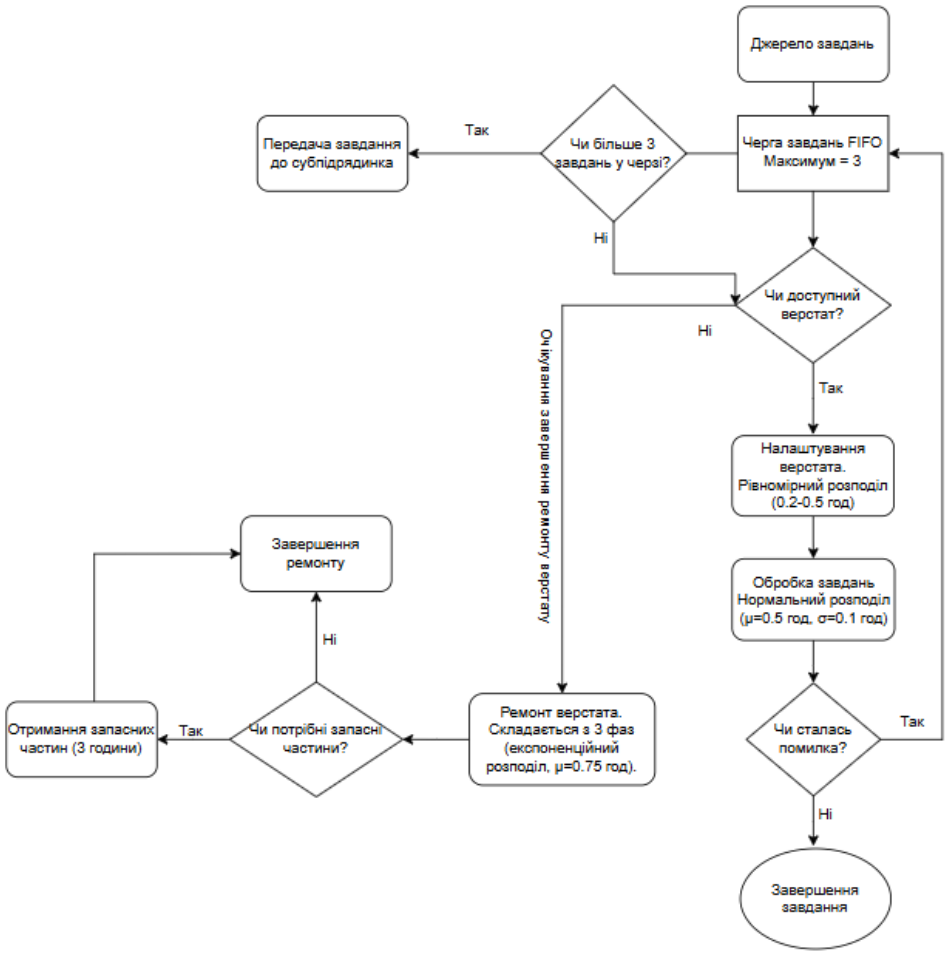


Рисунок 2.2 – Схематично зображене функціонування модифікованої системи

Модифікована модель дозволяє аналізувати вплив перевантаження системи та переривання ремонтних робіт на продуктивність і час обробки завдань.

Аналізуються ті ж цільові показники, але з урахуванням додаткових умов. Визначається вплив перенаправлення завдань до субпідрядника у разі переповнення черги при поломці та переривання ремонту через необхідність отримання запасних частин. Для завдань, переправлених до субпідрядника, розраховується частка таких завдань від загальної кількості. Також, розраховується частка часу переривання ремонту від загального часу ремонту, для більш детального оцінювання впливу додаткової умови на роботу системи.

Звичайна модель дозволяє оцінити базову продуктивність системи. Модифікована модель враховує всі умови завдання: вплив субпідрядника, переривання ремонту і їхні наслідки для системи.

2.3 Звичайний варіант формалізованої системи

Метою побудови імітаційної моделі системи є відображення роботи верстата. Для реалізації моделі використано концептуальний опис процесів обслуговування завдань, черг та поломок. Важливі характеристики системи включають інтенсивність надходження завдань, час виконання завдань, наладку верстата перед обробкою, а також поломки та їхній вплив на роботу верстата. Система моделюється на основі принципів теорії масового обслуговування з урахуванням випадкових величин, які впливають на її функціонування, таких як експоненційний, нормальний, рівномірний та ерланговий розподіли. Це дозволяє отримати показники завантаження верстата та статистичні оцінки часу виконання завдання, що є ключовими показниками ефективності системи.

Спершу, реалізуємо формальний опис фрагменту виконання завдань верстатом, які надходять зовні на виконання із $t = \exp(1)$. Маємо наступні етапи виконання завдання. СМО D_0. Прийняття завдання для верстата, із необмеженою

чергою та часом виконання t = 0, реалізовано задля збереження завдань у черзі перед початком виконання та контролю надходження завдань на подальшу обробку. Маємо блокування маршруту завдання на СМО D_1, наладку верстату. Умовами блокування маршруту є активний процес виконання завдання верстатом в СМО D_2 або активний процес наладки верстата в СМО D_1 або активний процес усунення поломки в СМО D_5. Ці умови дозволяють впевнитись в тому, що процес наладки не почнеться передчасно, при виконанні іншого завдання або при активному процесі усунення поломки у верстаті. Після розблокування маршруту завдання потрапляє у наступним етапом, а саме СМО D_1, наладку верстату перед виконанням завдання, що має часову затримку t = unif(0.2, 0.5). Наступний кроком, завдання переходить на етап виконання у СМО D_2 із часовою затримкою t = min{norm(0.5, 0.1),tnext(D4,t)-tcurr, що інтерпретується, як мінімальний час серед часу до виникнення поломки у верстаті або до часу успішного виконання завдання. Це дозволяє перервати завдання саме у момент виникнення поломки, завдання запам’ятовує прогрес свого виконання. При перериванні або завершенні виконання завдання прямує по одному з доступних маршрутів, один з яких є пріоритетним, але заблокованим, блокується він за відсутності активного процесу усунення поломки D_5, що свідчить про успішне завершення виконання завдання, як слідство виконане завдання покидає систему, N_1 – лічильник успішно виконаних завдань. В іншому випадку при перериванні виконання, завдання не є виконаним до кінця та прямує по пріоритетному маршруту у початок черги СМО D_0. та очікує на завершення ремонту і початку наладки перед продовженням виконання з моменту переривання виконання поломкою. Описаний вище(рис.2.2) фрагмент формалізованої моделі роботи верстату.

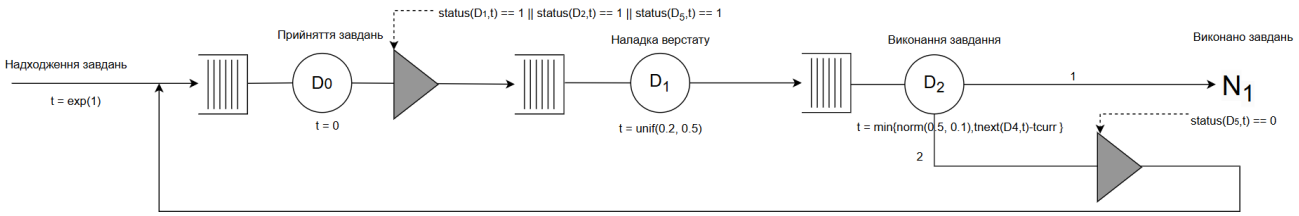


Рисунок 2.3 — Фрагмент формалізованої моделі роботи верстату, що описує процес виконання завдань верстатом

Опишемо другий фрагмент формалізованої моделі, що описує виникнення та процес усунення поломки верстату. Фрагмент реалізовано без зовнішнього надходження запитів, має одне завдання-поломку, що циркулює по ньому. За замовченням на початку моделювання завдання блокується на процесі D_3 із часовою затримкою t = 0. Це реалізовано із ціллю починати відлік часу для виникнення поломки з моменту, коли верстат працює над виконанням завдання. Розблокування маршруту до процесу виникнення поломки D_4 відбувається за умови, що процес виконання завдання верстатом D_2 є активним. Після розблокування маршруту завдання потрапляє в процес виникнення поломки із часовою затримкою t = norm(20,2), у випадковий момент часу трапляється поломка. Активна поломка може йти по одному з двох шляхів, більш пріоритетний шлях веде назад в чергу у СМО D_3, заблоковано при умові, що наладка D_1 не відбувається. Це реалізовано для запобігання початку процесу усунення поломки, при збіганні обставин, де процес виникнення поломки було запущено, завдання було виконано та виведено із системи, а поломка виникла при процесі наладки верстата для іншого завдання. Менш пріоритетним, але вільним маршрутом є маршрут, що веде до процесу усунення поломки із часовою затримкою t = erl(3,3/4), що інтерпретується як сума часу трьох ідентичних фаз усунення поломок із експоненційним законом розподілу із середнім значенням ¾. Після завершення обробки поломки даним процесом запит на виконання завдання циклічно повертається до процесу прийняття поломки D_3 та блокується на ньому, до виконання умов повторного розблокування маршруту. Описаний фрагмент(рис.2.3) формалізованої моделі роботи верстату.

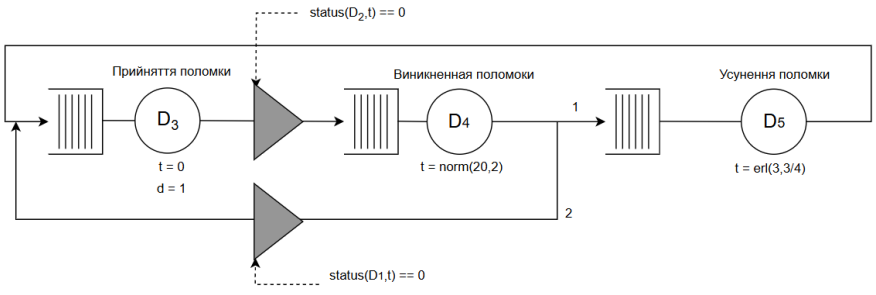


Рисунок 2.4 — Фрагмент формалізованої моделі роботи верстату, що описує процес виникнення та оброблення поломок верстату

Описана формалізована модель роботи верстату(рис 2.5).

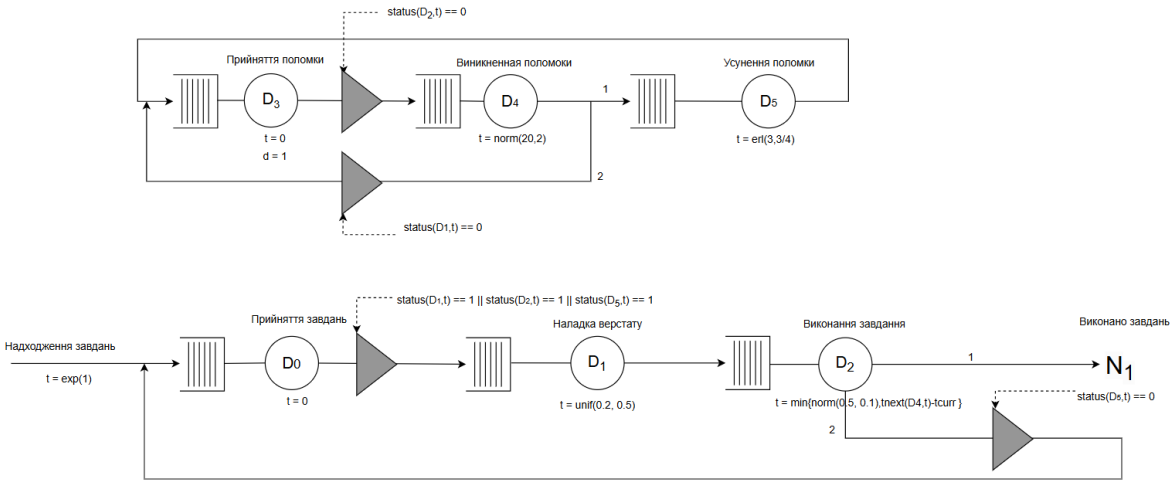


Рисунок 2.5 — Формалізована модель роботи верстату

Спершу, розрахуємо завантаження верстата, тобто частку часу, протягом якого верстат безпосередньо виконує завдання та перебуває у стані налаштування у процесах D_1 та D_2:

$$V_{зав} = \frac{T_1 + T_2}{T_{mod}} \quad (2.1)$$

де T_1 – час перебування всіх завдань на налаштуванні, T_2 – час перебування їх на виконанні, T_{mod} – загальний час моделювання.

Наступним кроком розрахуємо статистичні показники часу виконання завдання, тобто мінімальний, максимальний та середній час перебування завдання в системі, а також середньоквадратичне відхилення. Врахуємо всі етапи перебування завдання в системі від надходження до виходу. Завданнями, що на час завершення моделювання знаходились у системі знехтуємо, через достатньо велику кількість оброблених завдань системою, протягом моделювання. Маємо загальну кількість оброблених завдань системою записану у лічильник N_1. Припустимо, що в результаті моделювання $N_1 = i$. Тоді загальний час перебування всіх завдань в системі дорівнює $T = t_1 + \ldots + t_i$. Звідси маємо середній час перебування завдання в системі(2.2)

$$t_{сер} = \frac{T}{N1} = \frac{\sum_{k=1}^{N_1} t_k}{N1} \quad (2.2)$$

де T – загальний час перебування завдань у системі, N_1 – кількість завдань, що були виконані, t_k – час перебування k-го завдання в системі.

Мінімальний час перебування в системі(2.3)

$$t_{min}=\min(t_1, t_2 ,\ldots, t_i) \quad (2.3)$$

де $t1, t2 , \ldots , ti$ – часи перебування окремих завдань у системі, t_{min} – мінімальний час перебування завдання серед усіх.

Максимальний час перебування в системі(2.4)

$$t_{max}=\max(t_1, t_2 ,\ldots, t_i) \quad (2.4)$$

де $t1, t2 , \ldots , ti$ – часи перебування окремих завдань у системі, t_{max} – мінімальний час перебування завдання серед усіх.

Середньоквадратичне відхилення часу(2.5)

$$\sigma = \sqrt{\frac{1}{N_1}\sum_{k=1}^{N_1}(t_k - t_{сер})^2} \quad (2.5)$$

де t_k – час прибуття k-го завдання, $t_{сер}$ – середній час прибуття завдань, N_1 – кількість завдань, що розглядаються.

Після отримання цільових показників можна робити висновки, щодо продуктивності роботи системи.

2.4 Звичайний варіант формалізованої системи

Процеси та вихідні змінні, описані у підрозділі 2.1. залишаються аналогічними у модифікованій моделі. Нижче, буде наведено опис відмінностей модифікованої моделі від звичайного варіанту та додаткових умов врахованих в моделі.

Для аналізу системи в умовах змінених правил обслуговування завдань було запропоновано модифіковану модель. Основною відмінністю є введення додаткових умов, які враховують обмеження черги завдань під час поломки верстата, перенаправлення надлишкових завдань до субпідрядника та можливість переривання ремонту через необхідність отримання запасних частин. Буде досліджений, вплив цих змін на завантаження верстата та середній час виконання завдання.

Першою введеною умовою в систему є вилучення всіх, окрім останніх трьох завдань, в разі поломки, з черги очікування на виконання роботи верстатом, тобто із черги СМО D_0. Встановимо альтернативний маршрут із D_0 на вихід завдання із системи до субпідрядника N_2 та встановимо на ньому блокування, яке буде відкривати маршрут за умови, що верстат знаходиться у стані ремонту, тобто процес усунення поломки активний в СМО D_5 і водночас значення черги у СМО D_0 перевищило 3. Модифікований фрагмент зображено на рисунку 2.6.

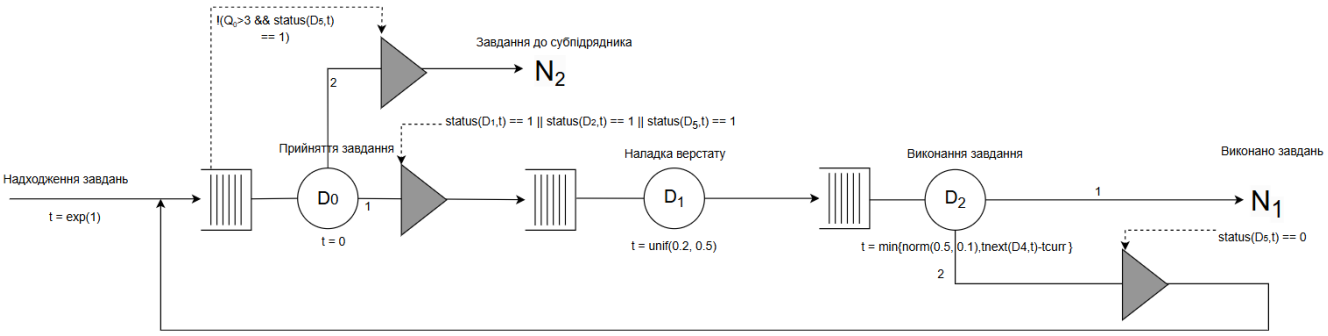


Рисунок 2.6 — Модифікований фрагмент формалізованої моделі роботи верстату, що описує процес обробки завдань верстатом

Наступною введеною умовою стане можливість переривання ремонту верстату задля пошуку запасних деталей. Для цього буде реалізовано третій фрагмент моделі, без прив'язки до стану ремонту верстату за умовою завдання, із надходженням зовнішніх запитів на пошук деталей з часовою затримкою $t = exp(100)$. У видку якщо під час виникнення запиту на пошук деталей ремонт не відбувався, нічого не буде траплятись. Інакше, при активному процесі пошуку деталі в СМО D_6 із детермінованим значенням часової затримки. $t = 3$, якщо починається ремонт верстату, то до його часової затримки, час виділений на ремонт верстату додається значення 3. Описаний фрагмент формалізованої моделі роботи верстату зображено на рисунку 2.7.

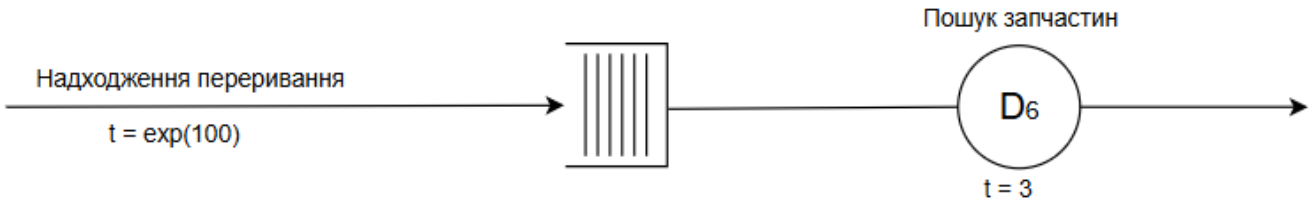


Рисунок 2.7 — Фрагмент формалізованої моделі роботи верстату, що описує процес пошуку деталей для верстату

Для забезпечення переривань під час ремонту було введено наступні зміни у фрагмент процесу ремонту верстату. У часову затримку процесу ремонту верстату D_5 було додано наступну умову $t = \begin{cases} erl\left(3, \frac{3}{4}\right), \text{якщо } status(D6, t) = 0 \\ erl\left(3, \frac{3}{4}\right) + 3, \text{якщо } status(D6, t) = 1 \end{cases}$, що перевіряє перед початком процесу ремонту чи почався процес пошуку деталей і чи потрібні запасні деталі для ремонту, при позитивній відповіді до розрахованої часової затримки процесу ремонту верстату додається детерміноване значення 3, що імітує часову затримку на 3 години для пошуку запасних частин. Модифікований фрагмент формалізованої моделі зображено на рисунку 2.8.

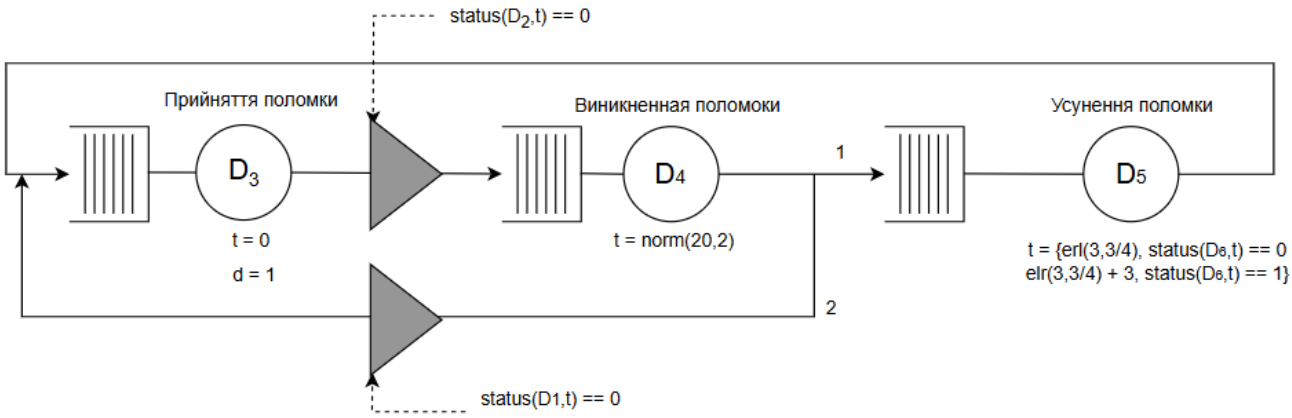


Рисунок 2.8 — Модифікований фрагмент формалізованої моделі роботи верстату, що описує процес ремонту верстату

Описана модифікована формалізована модель роботи верстату зображена на рисунку 2.9.

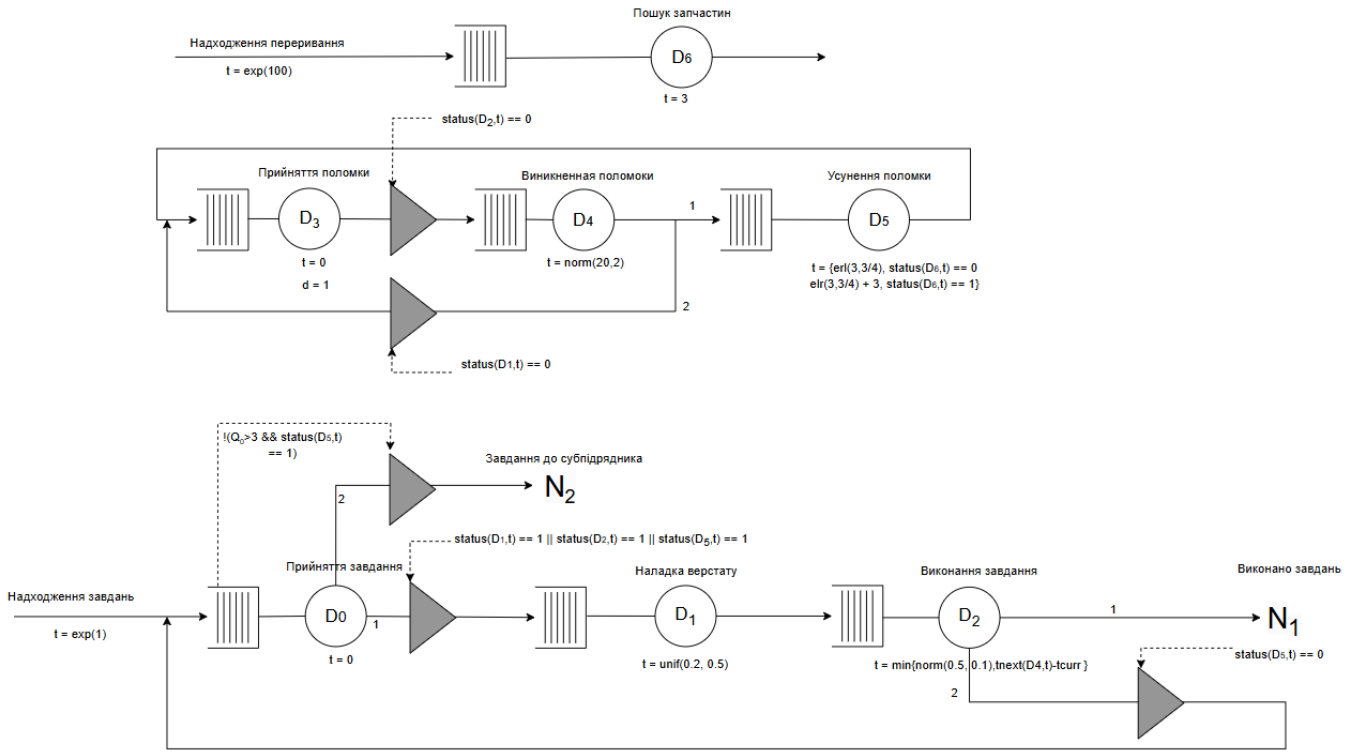


Рисунок 2.9 — Модифікована формалізована модель роботи верстату

Нововведені умови можуть мати вплив на цільові змінні описані в пункті 2.1, що буде досліджено далі в роботі. Час перебування завдання в системі, так само розраховується на запитах оброблених системою, не враховуючи не завершені

задачі, а також задачі переведені на субпідрядника, описані за формулами у пункті 2.3.

Після отримання цільових показників модифікованої моделі та їх аналізу, можна буде зробити висновки, щодо продуктивності роботи модифікованої системи та загального впливу модифікацій на систему.

Для більш наочного розуміння впливу модифікації на модель, було введено додаткові вихідні змінні у модифіковану модель: Частка завдань, переданих на субпідрядника. Кількість завдань, переправлених через перевантаження системи під час поломки верстата розділена на загальну кількість завдань, що надійшли у систему, що дозволяє оцінити ступінь використання субпідрядника. Частка часу витраченого на пошук запасних деталей, до загального часу ремонту верстата, що дозволяє оцінити вплив переривань на ремонт і відповідно, на завантаження верстату.

Для розрахунку частки завдань, переданих на субпідрядника, протягом моделювання поділимо кількість завдань, що надійшли до субпідрядника, тобто N_2 на загальну кількість завдань оброблених системою, тобто $N_1 + N_2$. (2.6)

$$\text{Частка}_{\text{завдань}} = \frac{N_2}{N_1 + N_2} \tag{2.6}$$

Для розрахунку частки часу витраченого на пошук запасних деталей розділимо час витрачений на пошук запасних деталей $T_{\text{пошуку}}$ на загальний час ремонту верстату $T_{\text{ремонту}}$, який включає в себе і час переривання ремонту для здійснення пошуку. (2.7)

$$\text{Частка}_{\text{завдань}} = \frac{\text{Тпошуку}}{\text{Тремонту.}} \tag{2.7}$$

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ

3.1 Аргументація обраних засобів реалізації

Алгоритм імітації побудований за принципом просування часу до найближчої події. Цей підхід забезпечує оптимальну продуктивність і точність, уникаючи непродуктивних кроків, які виникають при використанні дискретного кроку часу. Замість цього модель переходить від однієї події до іншої, використовуючи часові мітки для управління перебігом симуляції. Також, існуючим підходом є послідовне проведення об’єктів вздовж моделі, коли кожне завдання або елемент системи поступово проходить усі етапи обробки. Однак цей метод менш ефективний для стохастичних систем із динамічними подіями, оскільки він не дозволяє оптимально відслідковувати та обробляти події, що відбуваються у різних частинах системи одночасно. Крім того, у складних моделях, де є кілька черг або механізмів маршрутизації (як у нашій системі із поломками та поверненням завдань у чергу), такий підхід ускладнює реалізацію логіки залежностей між подіями. Тому було обрано саме метод просування часу до найближчої події, який краще підходить для дискретно-подвійного моделювання з високою точністю та ефективністю.

Крім того, реалізовано просування стану моделі в часі за подіями. Це дозволяє відстежувати стан системи й ефективно обробляти складні сценарії з динамічними подіями, підтримуючи високу точність та адаптивність моделі. Ще одним альтернативним підходом є процесно-орієнтований спосіб, який може бути ефективним для систем із чітко визначеними послідовностями дій, але не в разі складних систем із великою кількістю залежностей.

Особливу увагу приділено обробці певних подій, що відбуваються одночасно: порядок їх виконання визначено відповідно до пріоритетів, щоб забезпечити коректність роботи системи. А саме, при одночасній події поломка

верстата і завершення завдання, першою обробляється подія виникнення поломки, від якої залежить маршрут виходу завдання з верстату, після чого відбувається завершення виконання завдання верстатом та вихід для повернення недробленого завдання до початку черги.

Для реалізації імітаційної моделі системи було обрано Java завдяки її високій продуктивності та підтримці об'єктно-орієнтованого підходу.

Стохастичні характеристики системи, включно з експоненціальними, нормальними, рівномірними та ерланговими розподілами часу, моделювалися за допомогою генераторів випадкових чисел, доступних у Java. Це дозволяє максимально точно відтворити реальні умови роботи системи.

Для збору інформації під час виконання моделі було застосовано статистичні методи, які є найбільш ефективними для аналізу складних стохастичних систем. Ці методи дозволяють накопичувати дані про час виконання завдань, рівень завантаження верстата та інші показники. Результати, отримані за допомогою статистичних методів, є точними та обґрунтованими, що важливо для аналізу та вдосконалення роботи системи. Аналітичні методи не підходять для розв'язання задачі через складність динамічної взаємодії компонентів системи та наявність блокування маршрутів. Поломки верстата, повернення завдань у початок черги й інші події впливають на зміну стану системи в реальному часі. Аналітичний підхід не може врахувати ці зміни, оскільки вони є стохастичними й нелінійними. Наприклад, у ситуаціях, коли поломка блокує маршрут завдань, потрібно враховувати динамічну зміну черги, що виходить за межі можливостей аналітичного моделювання.

Обрані методи забезпечують високу точність моделювання, мінімізуючи похибки, пов’язані з округленням часу або спрощенням розподілів. Застосування статистичних методів дозволяє моделювати систему максимально наближено до реальних умов, що є важливим для розв’язання поставленої задачі

3.2 Опис програмної реалізації імітаційної моделі

Загальний псевдокод для програмної реалізованого імітації моделі може виглядати наступний чином:

Input variables X, input parameters P, current time t = 0, modeling time Tmod

Initialize current state S

while t < Tmod:

1. Find the nearest event (tNext) across all elements

2. Update statistics for output variables Y

3. Update current time t := tNext

4. Call the state transformation function E(S, t) for the nearest event

5. Move the state of the model S := E(S, t) (change model state based on the event)

6. Process the actions triggered by the event (e.g., tasks and routing to other elements)

7. Recalculate state of all elements affected by the current event

8. Print model information after each event

End while

Calculate final statistics Y (Workload, Statistics of task time in system)

Далі буде описано основні деталі програмної реалізації базової та модифікованої імітаційних моделей.

Для реалізації моделі було використано універсальний алгоритм імітації моделі масового обслуговування з багатоканальним обслуговуванням, з вибором маршруту за пріоритетом. Універсальні класи, чию поведінку треба було змінити задля виконання всіх умов моделювання та належного функціонування моделі, було наслідувано та модифіковано належним чином.

Універсальний клас Model.java реалізує основну логіку симуляції системи масового обслуговування, управляючи взаємодією між елементами моделі і забезпечуючи правильну послідовність обробки подій. Клас містить масив елементів системи, синхронізує час подій, визначає наступну подію, обробляє

блокування маршрутів і оновлює статистику роботи системи. Основний метод simulate реалізує цикл симуляції, який виконується до заданого часу, обробляючи події за їхнім хронологічним порядком, оновлюючи стан елементів і виконуючи дії, пов’язані з подіями. Для кожного елемента збирається статистика, яка виводиться у фіналі симуляції через метод printResult, включаючи середню довжину черги, завантаженість процесорів і ймовірність відмов. Клас є універсальним ядром, яке дозволяє моделювати різноманітні сценарії масового обслуговування.

Наступний клас ModelWorkTable розширює базовий клас Model для реалізації моделі роботи верстата. Він додає детальну статистику виконання завдань, особливу увагу приділено обчисленню цільових показників: завантаженість верстату, середній, мінімальний і максимальний час перебування завдань у системі, стандартне відхилення.

Також збирає дані про загальну кількість оброблених завдань. Додаткові методи дозволяють аналізувати завантаженість процесів налаштування та обробки, тривалість вирішення поломок і неефективного часу простою системи. Клас також перевизначає метод симуляції, додаючи логіку вибору подій із однаковим часом виконання при певних умовах за пріоритетами та відображенням додаткової інформації. У результатах симуляції виводяться основні показники, такі як завантаження верстату та статистика часу виконання завдань. Також для більш детального аналізу функціонування моделі виведено додаткову інформацію в результати симуляції. Цей клас забезпечує комплексний аналіз роботи моделі для завдань виробничого середовища.

```java
public class ModelWorkTable extends Model {
    private double totalSetupTime;
    private double totalProcessingTime;

    public ModelWorkTable(Element...elements) {
        super(elements);
        this.totalSetupTime = 0;
        this.totalProcessingTime = 0;
    }

    // Inner class to encapsulate task processing statistics
    private static class TaskStatistics {
        final double avgTimeInSystem;
        final double minTimeInSystem;
        final double maxTimeInSystem;
        final double standardDeviation;
        final int totalProcessedTasks;

        TaskStatistics(double avgTimeInSystem, double minTimeInSystem,
                       double maxTimeInSystem, double standardDeviation,
                       int totalProcessedTasks) {
            this.avgTimeInSystem = avgTimeInSystem;
            this.minTimeInSystem = minTimeInSystem;
            this.maxTimeInSystem = maxTimeInSystem;
            this.standardDeviation = standardDeviation;
            this.totalProcessedTasks = totalProcessedTasks;
        }
    }
```

Рисунок 3.1 — Фрагменту коду класу ModelWorkTable

Task реалізує клас, який представляє завдання в мережі масового обслуговування. Кожне завдання має унікальний ідентифікатор, час входу в систему та час виходу з системи. Час виходу можна змінювати, що дозволяє відстежувати повний цикл обробки завдання. Унікальний ідентифікатор автоматично генерується за допомогою статичного лічильника nextId, забезпечуючи унікальність кожного створеного екземпляра. Клас слугує універсальною сутністю для обробки завдань у моделі.

```java
public class Task {
    private static int nextId = 0;
    private final int id;
    private final double timeIn;
    private double timeOut;

    public Task(double timeIn) {
        this.timeIn = timeIn;
        this.timeOut = timeIn;
        this.id = nextId;
        nextId++;
    }

    public int getId() {
        return id;
    }

    public double getTimeIn() {
        return timeIn;
    }

    public double getTimeOut() {
        return timeOut;
    }

    public void setTimeOut(double timeOut) {
        this.timeOut = timeOut;
    }
}
```

Рисунок 3.2 — Фрагменту коду класу Task

VerstatTask розширює базовий клас Task і додає додаткові характеристики для завдань, які можуть бути перервані. Нові поля включають прапорець isInterrupted для позначення стану переривання завдання, fullTime для збереження повного часу, необхідного для виконання завдання, та leftTime для відстеження залишкового часу на момент переривання. Цей клас дозволяє моделювати більш складні сценарії, де завдання можуть зупинятися й продовжуватися, а також зберігати інформацію про прогрес виконання екземпляра.

```java
public class VerstatTask extends Task {
    private boolean isInterrupted;
    private double fullTime;
    private double leftTime;

    public VerstatTask(double timeIn) {
        super(timeIn);
        this.isInterrupted = false;
        this.fullTime = 0.0;
        this.leftTime = 0.0;
    }

    public void markInterrupted() {
        this.isInterrupted = true;
    }

    public boolean isInterrupted() {
        return isInterrupted;
    }

    public double getFullTime() {
        return fullTime;
    }

    public double getLeftTime() {
        return leftTime;
    }

    public void setFullTime(double fullTime) {
        this.fullTime = fullTime;
    }

    public void setLeftTime(double leftTime) {
        this.leftTime = leftTime;
    }
}
```

Рисунок 3.3 — Фрагменту коду класу VerstatTask

CreateVerstatTasks визначає спеціалізований клас для створення завдань типу VerstatTask в системі. Він успадковує функціональність від універсального класу Create і перевизначає метод createTask, щоб створювати об'єкти типу VerstatTask, які підтримують додаткові властивості, такі як переривання, залишковий час та повний час виконання.

```
public class CreateVerstatTasks extends Create {
    public CreateVerstatTasks(String name, double delay, double initialTNext) {
        super(name, delay, initialTNext);
    }

    @Override
    protected Task createTask() {
        return new VerstatTask(super.getTCurr());
    }
}
```

Рисунок 3.4 — Фрагмент коду класу CreateVerstatTasks

Process, також наслідує Element, реалізує універсальний клас, що моделює елемент обслуговування завдань у мережі масового обслуговування. Він містить чергу для завдань і множину каналів, які можуть обробляти ці завдання паралельно. Клас забезпечує управління чергою, розподіл завдань на вільні канали, обробку вихідних завдань і передачу їх наступним елементам системи через маршрути. Крім того, клас дозволяє відстежувати статистику, таку як середній розмір черги, час роботи каналів і кількість невиконаних завдань через переповнення черги. Реалізація підтримує багатоканальну обробку завдань і дозволяє моделювати процеси з обмеженим розміром черги та змінною кількістю каналів. Код даного класу є надто об’ємним, його можна переглянути у додатку А.

Універсальний клас Route описує шлях, яким завдання можуть рухатись від одного елементу до іншого в процесі обробки. Він зберігає інформацію про елемент, через який проходить маршрут, ймовірність цього шляху, пріоритет та можливість блокування маршруту через умови, визначені функцією Block. Клас підтримує кілька конструкторів для гнучкого налаштування маршрутів. Метод isBlocked перевіряє, чи заблокований маршрут для конкретного завдання за допомогою функції блокування, яка може визначати, чи можна передати завдання через цей маршрут. Функція блокування реалізується через функціональний інтерфейс Block, що дозволяє задавати умови блокування для кожного конкретного випадку.

```java
public class Route {
    private final Element element;
    private int priority = 0;
    private double probability = 1.0;
    private Block block = null;

    public Route(Element element) {
        this.element = element;
    }

    public Route(Element element, Block block) {
        this.element = element;
        this.block = block;
    }

    public Route(Element element, double probability) {
        this.element = element;
        this.probability = probability;
    }

    public Route(Element element, double probability, int priority) {
        this.element = element;
        this.probability = probability;
        this.priority = priority;
    }

    public Route(Element element, double probability, int priority, Block block) {
        this.element = element;
        this.probability = probability;
        this.priority = priority;
        this.block = block;
    }

    public boolean isBlocked(Task task) {
        if (block == null) {
            return false;
        }
        try {
            return block.call(task);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

Рисунок 3.5 — Фрагмент коду класу Route

Dispose розширює клас Element і призначений для обробки та збереження завдань, що проходять через цю одиницю. Він має список processedTasks, в якому зберігаються оброблені завдання. Метод inAct викликає метод батьківського класу для виконання основних дій, додає оброблене завдання до списку processedTasks і викликає метод outAct, щоб завершити обробку завдання. Метод printInf виводить інформацію про кількість завдань, що пройшли через цей елемент. Метод getProcessedTasks повертає список всіх оброблених завдань для подальшого аналізу.

```
public class Dispose extends Element {
    ArrayList<Task> processedTasks = new ArrayList<>();

    public Dispose(String name) {
        super(name);
    }

    @Override
    public void inAct(Task task) {
        super.inAct(task);
        processedTasks.add(task);
        super.outAct();
    }

    @Override
    public void printInfo() {
        System.out.println(getName() + " quantity = " + getQuantity());
    }

    public ArrayList<Task> getProcessedTasks() {
        return processedTasks;
    }
}
```

Рисунок 3.6 — Фрагмент коду класу Dispose

Клас FunRand містить методи для генерації випадкових значень, що відповідають різним розподілам ймовірностей. Метод Exponential генерує випадкове значення з експоненціальним розподілом з середнім значенням. Метод Uniform генерує значення, що підпорядковується рівномірному розподілу на інтервалі. Метод Normal генерує випадкове значення, що відповідає нормальному розподілу з середнім значенням та стандартним відхиленням. Метод Erlang генерує випадкове значення за ерланговим розподілом, що базується на кількості експоненційних розподілів та середньому значенні. Усі методи використовують стандартні математичні операції для генерації випадкових значень.

```java
public class FunRand {
    /**
     * Generates a random value according to an exponential
     * distribution
     *
     * @param delayMean mean value
     * @return a random value according to an exponential
     * distribution
     */
    public static double Exponential(double delayMean) {
        double a = 0;
        while (a == 0) {
            a = Math.random();
        }
        a = -delayMean * Math.log(a);
        return a;
    }

    /**
     * Generates a random value according to a uniform
     * distribution
     *
     * @param min min value
     * @param max max value
     * @return a random value according to a uniform distribution
     */
    public static double Uniform(double min, double max) {
        double a = 0;
        while (a == 0) {
            a = Math.random();
        }
        a = min + a * (max - min);
        return a;
    }
    /**
     * Generates a random value according to a normal (Gauss)
     * distribution
     *
     * @param delayMean      mean value
     * @param delayDev standard deviation of normal distribution
     * @return a random value according to a normal (Gauss) distribution
     */
    public static double Normal(double delayMean, double delayDev) {
        double a;
        Random r = new Random();
        a = delayMean + delayDev * r.nextGaussian();
        return a;
    }
    /**
     * Generates a random value according to an Erlang distribution
     *
     * @param times k (integer)
     * @param delayMean  lambda
     * @return a random value according to an Erlang distribution
     */
    public static double Erlang(double times, double delayMean) {
        double product = 1.;
        for (int i = 0; i < (int) times; ++i) {
            double a = Math.random();
            while (a == 0) {
                a = Math.random();
            }
            product *= a;
        }
        return -Math.log(product) / delayMean;
    }
}
```

Рисунок 3.7 — Фрагмент коду класу FunRand

Функція worktable ініціалізує та налаштовує елементи моделювання для системи. Створення завдань для верстату create, обробка в черзі waitingTask,

налаштування верстату setupTask, обробка завдань верстатом solvingTask, їх вихід із системи dispose. Також ініціалізовано елементи фрагменту виникнення та обробки поломок верстату, де буде циркулювати заздалегідь встановлене, єдине завдання поломки верстату. Всі елементи підключаються один до одного через маршрути, які визначають порядок виконання та умови слідування і блокування для кожного етапу. Кожному процесу задано свій розподіл часу для виконання. Моделювання відбувається через створення екземпляра ModelWorkTable, в якому виконується симуляція системи протягом 1000 годин.

3.3 Опис програмної реалізації модифікованої імітаційної моделі

У роботі розроблено модифіковану імітаційну модель із введенням додаткових умов. Модель включає розширення стандартної імітаційної моделі, що дозволяє визначити вплив даних умов на цільові показники моделювання, а також враховувати специфічні аспекти, такі як пропорція підрядних завдань до загально оброблених моделюю і частку часу витраченого на пошук деталей для процесу ремонту верстату. За допомогою модифікацій було реалізовано додаткові метрики, зокрема, обчислення кількості переривань та їхнього впливу на загальний час обробки. Результати симуляції надають детальну статистику про ефективність роботи системи, що дозволяє оцінити її продуктивність і виявити можливі проблеми або переваги, пов'язані з переправленням завдань на підрядника.

Модифікована версія основного класу імітаційної моделі ModelWorkTableMod включає додаткові функції для обчислення та аналізу субпідрядних завдань і переривань, при поломці верстата, у системі. Основною зміною є додавання параметрів для відслідковування пропорції субпідрядних завдань до загальної кількості оброблених завдань, а також часу, витраченого на пошук деталей при перериванні ремонту верстату. У порівнянні зі стандартною моделлю, ця версія дозволяє ефективно оцінювати додаткові метрики, що є

важливими для врахування при введені додаткових умов у модель. В результаті, ця модель дає точне уявлення про роботу системи в нових умовах з перериваннями ремонту верстату та переведенням завдань на субпідрядника при перевантаженні черги під час ремонту верстату.

Модифікована версія процесу Process0Mod відрізняється від звичайної версії Process0WaitingTask кількома важливими змінами, які враховують обробку завдань у разі поломки верстата та необхідності розвантаження системи за допомогою субпідрядників. У модифікованій версії додано поле solvingBreakdown, яке представляє процес усунення поломки верстата. Це дозволяє враховувати ситуацію, коли верстат ламається. Якщо процес усунення поломки активний (його стан дорівнює 1), завдання в черзі передаються до субпідрядника, якщо кількість завдань у черзі перевищує певний ліміт (3 завдання). Завдання, які не входять до цієї кількості, для розвантаження системи, передаються на обробку субпідряднику в порядку черги разом з необробленими завданнями, які встали в початок черги при перериванні роботи верстату, через метод removeTasksForSubcontractor. У моделі також додано поле disposeSubcontractor, яке вказує на об'єкт Dispose, що відповідає за обробку завдань субпідрядником. Якщо черга до верстата містить більше ніж 3 завдання, всі зайві завдання починаючи з початку черги передаються на обробку до субпідрядника. Як і в звичайній версії, завдання, яке було перервано, ставиться на початок черги. У модифікованій версії ця логіка також зберігається, але додається перевірка на те, чи є активний процес поломки, що впливає на подальшу обробку черги.

```
public class Process0Mod extends Process0WaitingTask {
    private Process solvingBreakdown = null;
    private Dispose disposeSubcontractor = null;

    public Process0Mod(String name, double delayMean) {
        super(name, delayMean);
    }

    public void setSolvingBreakdown(Process solvingBreakdown) {
        this.solvingBreakdown = solvingBreakdown;
    }

    public void setDisposeSubcontractor(Dispose disposeSubcontractor) {
        this.disposeSubcontractor = disposeSubcontractor;
    }

    public void removeTasksForSubcontractor(int maxRemainingTasks, Dispose subcontractorDispose) {
        while (queue.size() > maxRemainingTasks) {
            Task task = queue.poll();
            if (task != null) {
                subcontractorDispose.inAct(task); // Передаємо завдання на субпідрядника
            }
        }
    }

    @Override
    public void inAct(Task task) {
    super.inAct(task);
        // Логіка активації очищення черги
        if (solvingBreakdown.getState() == 1) { // Якщо процес 5 активний (поломка)
            removeTasksForSubcontractor(3, disposeSubcontractor); // Тільки останні 3 завдання залишаються
        }
    }
}
```

Рисунок 3.8 — Фрагмент коду модифікованої версії процесу Process0Mod

Модифікація класу Process5SolvingBreakdown впроваджує кілька важливих змін у порівнянні з універсальним класом Process, що використовується у базовій моделі для моделювання процесу обробки поломки верстата. Основна мета цієї модифікації — врахувати переривання на пошук запчастин для процесу ремонту верстата під час поломки. У цій модифікації додано поле searchProcess, яке вказує на процес пошуку деталей для ремонту. Це дозволяє визначити, чи буде процес ремонту активним або перерваним через брак деталей. Якщо процес пошуку запчастин пов'язаний із ремонтом верстату не активний, то час, необхідний для усунення поломки, буде розраховано за ерланговим розподілом розподілом. У випадку, коли процес ремонту переривається через відсутність деталей, час переривання, а саме три години, додається до запланованого часу, і це викликає збільшення лічильника переривань interrupts. Метод inAct відповідає за обробку завдань, які поступають у процес ремонту. Якщо вільний канал доступний,

завдання додається в нього, а час наступної події обчислюється на основі того, чи активний процес пошуку запасних деталей. Якщо процес пошуку активний, додається час переривання із фіксованим значенням три до часу розрахованого на ремонт верстату, що дозволяє змоделювати переривання ремонту на пошуку запасних деталей при їх нестачі. Метод startNextTasks обробляє завдання з черги, коли вільний канал доступний. Для кожного завдання перевіряється стан процесу пошуку: якщо він активний, час ремонту верстату обчислюється згідно з урахуванням затримки також фіксується в лічильнику переривань, а якщо ні і переривання не відбувається, то час процесу ремонту верстату обчислюється без врахування переривань. Ці зміни дозволяють моделювати ситуацію, коли процес ремонту верстату може бути перерваний через нестачу запасних частин, що впливає на час вирішення поломки та загальну ефективність моделі.

Основні функції worktable та worktablemod мають багато спільного, оскільки обидві визначають елементи моделі та маршрути для виконання завдань. Однак є кілька ключових відмінностей між звичайною моделлю та модифікованою, які пов'язані з відправкою завдань до субпідрядника за визначених раніше умов та переривання ремонту верстату для пошуку запчастин.

У модифікованій моделі замість Process0WaitingTask використовується Process0Mod, що містить додаткову логіку для переведення завдань з черги за принципом FIFO на субпідрядника при поломці верстату та одночасного перевантаження черги, кількість завдань більше трьох, три останні лишаються на обробку самим верстатом після поломки. Для цього додано елемент disposeSubcontractor, що обробляє завдання, які передаються субпідряднику.

В звичайній моделі поломка верстату обробляється через процес, що має стандартний алгоритм для обробки поломок. В модифікованій додано нову логіку, пов'язану з перериваннями при ремонті верстату. Для цього додаються нові процеси, а саме createInterrupt, що генерує переривання, і search, який моделює процес пошуку запчастин і від якого залежить час обробки поломок процесом solvingBreakdown.

У модифікованій моделі маршрути включають нові умови для передавання завдань до субпідрядника при поломці верстату, якщо кількість завдань у черзі перевищує три, а також маршрут для обробки переривань, пов'язаних з ремонтом верстату.

3.4 Аргументація обраних засобів реалізації

Аргументація обраних засобів реалізації є важливим етапом у процесі побудови імітаційної моделі, оскільки від правильного вибору інструментів, методів та технічних рішень залежить не лише точність моделювання, а й можливість подальшого аналізу результатів, масштабування моделі, адаптації до змінних умов тощо. У контексті даної роботи під засобами реалізації слід розуміти вибір мови програмування, структур даних, алгоритмів моделювання, а також підхід до організації симуляційного середовища.

Обґрунтування таких рішень дозволяє впевнено стверджувати, що побудована модель не лише відповідає теоретичному опису, але й коректно функціонує на практиці, демонструючи стабільну поведінку при різних налаштуваннях і параметрах. Аргументація вибору інструментів також дає змогу іншим дослідникам або фахівцям, які потенційно можуть працювати з моделлю, краще зрозуміти логіку реалізації та відтворити її в подальших дослідженнях.

Під час реалізації моделі особливу увагу приділялось методу просування часу до найближчої події (event-driven simulation), який є одним із найбільш ефективних для дискретно-подійного моделювання. Такий підхід дозволяє уникнути зайвих обчислень та зменшує навантаження на ресурси, оскільки модель не симулює події в ті моменти часу, коли в системі нічого не відбувається. Саме тому було обрано мову програмування Java, яка надає достатню гнучкість, високу продуктивність та зручні інструменти для реалізації об’єктно-орієнтованої архітектури.

Для підтвердження того, що модель реалізована правильно, проводиться

верифікація — тобто перевірка її поведінки в умовах базових параметрів. Верифікація дозволяє пересвідчитись у тому, що модель дає логічні та очікувані результати, коректно реагує на зміну вхідних даних та не містить критичних помилок у реалізації логіки подій.

Наведено результати верифікації (рис. 3.9), де зафіксовано значення основних вихідних змінних та проаналізовано реакцію системи на зміну ключових параметрів. Отримані дані свідчать про те, що реалізована модель функціонує відповідно до заданої концепції та адекватно відображає поведінку виробничого процесу в умовах впливу стохастичних чинників.

	Вхідні змінні									Вихідні змінні					Додаткові вихідні змінні						
Прогін	Затримка CREATE створення завдання, год	Затримка PROCESS 1 наладка верстату , год		Затримка PROCESS 2 обробка завдання, год		Затримка PROCESS 4 виникнення поломки, год		Затримка PROCESS 5 усунення поломки, год		Завантаження верстату	Мінімальний час виконання завдання, год	Середній час виконання завдання, год	Максимальний час виконання завдання, год	Квадртичне відхилення часу виконання завд	Сумарний час наладки верстату, год	Сумарний час обробки завдання верстатом, го	Сумарний час ремонту верстату, год	Сумарний час відсутності завдань для верстат	Кількість надійдених завдань у систему	Кількість оброблених завдань системою	Кількість завдань у системі при стопі симуляції
№	exp	unif, min	unif, max	norm, mean	norm, dev	norm, mean	norm, dev	elr, n	elr, mean												
1	1	0,2	0,5	0,5	0,1	20	2	3	0,75	0,86	0,59	10,38	31,31	7,6	359,44	503,98	85,87	50,89	999	996	3
2	**0.1**	0,2	0,5	0,5	0,1	20	2	3	0,75	0,87	1	454,4	899,93	262,12	367,83	507,11	125,06	0	9986	1014	8972
3	1	0,2	**0,22**	0,5	0,1	20	2	3	0,75	0,71	0,42	2,82	11,48	2,33	214,04	496,03	106,08	183,91	1001	998	3
4	1	0,2	0,5	**0,2**	0,1	20	2	3	0,75	0,59	0,24	1,99	10,69	1,93	361,42	236,66	85,5	316,63	1026	1023	3
5	1	0,2	0,5	0,5	0,1	**5**	2	3	0,75	0,73	0,6	90,96	205,22	50,06	314,52	412,89	270,21	2,69	1021	821	200
6	1	0,2	0,5	0,5	0,1	20	2	**1**	0,75	0,84	0,49	3,59	19,21	2,61	349,89	492,44	42,62	115,18	987	981	6

Рисунок 3.9 — Результати верифікації моделі

На верифікації (рис.3.9) виділено цільові вихідні показники, а також наведено додаткові вихідні змінні, для остаточного розуміння того, що модель відпрацьовує правильно та належним чином реагує на зміну вхідних параметрів.

Після першого прогону із параметрам за замочуванням, що були дані в умові, фіксуємо значення вихідних змінних. Показники відповідають нормі, імітація моделі відпрацьовую нелажним чином та видає очікувані статистичні показники.

За результатами другого прогону при зменшенні середнього значення часової затримки генерації нових запитів на виконання верстатом з 1 до 0.1, бачимо наступні результати. Завантаження верстату не значно збільшено. Час виконання завдання за всіма статистичними показниками збільшився. Такі показники обумовлено більшою кількістю завдань і є цілком очікуваними, модель реагує

належним чином. При аналізі додаткових вихідних змінних не виявлено аномальної поведінки моделі, дані відповідають нормі. Можна зазначити, що при додатковому навантаженні на модель, немає відсутності запитів на обробку та відповідно час простою верстату без ремонту, наладки та виконання нульовий, що також є очікуваним та логічним результатом.

За результатами третього прогону при зменшенні максимального значення часової затримки наладки верстата перед виконанням запитів 0.5 до 0.22, бачимо наступні результати. Завантаження верстату зменшилось через менший час наладки, а відповідно і швидший час обслуговування запиту. Час виконання завдання за всіма статистичними показниками зменшився, через те, що верстата налаштовується швидше, відповідно і швидше обробляються запити та виходять із системи. Це є очікуваним і логічним результатом. При аналізі додаткових вихідних змінних не виявлено аномальної поведінки моделі, дані відповідають нормі. Загальний час наладки верстату значно зменшився, що також є очевидною реакцією вихідних даних. Загальний час простою верстату збільшився, що є логічним.

За результатами четвертого прогону при зменшенні середнього значення часової затримки виконання запитів верстатом з 0.5 до 0.2, аналогічні результати. Завантаження верстату значно зменшилось. Час виконання завдання за всіма статистичними показниками також зменшено. Такі показники обумовлено більшою швидкістю обробки верстатом завдань і є логічними. При аналізі додаткових вихідних змінних не виявлено аномальної поведінки моделі, дані відповідають нормі. Загальний час виконання завдань верстатом зменшився.

За результатами п’ятого прогону при зменшенні середнього значення часової затримки виникнення поломок верстату з 20 до 5, бачимо наступні результати. Завантаження верстату зменшилось. Мінімальний час виконання завдання залишається стабільним, а середнє і максимальне значення збільшені, і відповідно середньоквадратичне. Такі показники обумовлено більшою частотою виникання поломко на верстаті і це є цілком очікуваними, модель реагує належним чином. При аналізі додаткових вихідних змінних не виявлено аномальної поведінки

моделі, дані відповідають нормі. Загальний час ремонту значено збільшився, що є логічним.

За результатами шостого прогону при зменшенні кількості етапів усунення поломки на верстаті з 3 до 1, а відповідно і її часової затримки, бачимо наступні результати. Завантаження верстату є стабільним. Час виконання завдання за всіма статистичними показниками значно зменшився. Такі показники обумовлено меншою кількістю часу, що витрачається на ремонт верстату і є очікуваними, модель реагує належним чином. При аналізі додаткових вихідних змінних не виявлено аномальної поведінки моделі, дані відповідають нормі. Загальний час ремонту верстату значно зменшився, що відповідає логіці роботи моделі в даних умовах.

Після запуску імітаційної моделі отримуємо логи із описом роботи процесів моделі та результати, наведені на рисунку 3.10

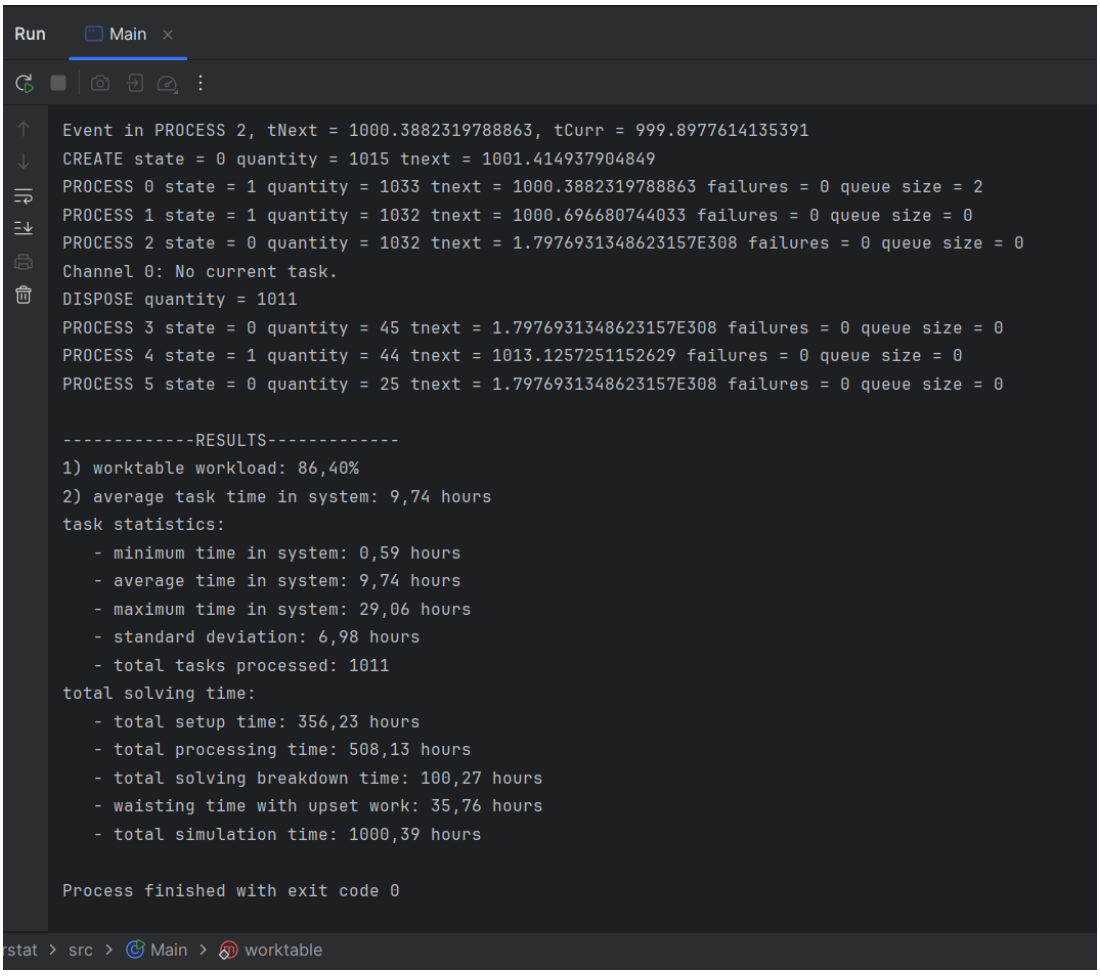


Рисунок 3.10 — Результат виконання коду імітаційної моделі

Як бачимо реалізація імітаційної моделі дозволяє нам наочно переконатись у правильності функціонування всіх процесів моделі. Перевіримо основні умови задачі.

Завдання поступило на етап наладки верстата:

```
CREATE state = 0 quantity = 1015 tnext = 1001.414937904849
PROCESS 0 state = 1 quantity = 1032 tnext = 999.8378461222414 failures = 0 queue size = 3
PROCESS 1 state = 1 quantity = 1031 tnext = 999.8977614135391 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 1031 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 1010
PROCESS 3 state = 0 quantity = 45 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 44 tnext = 1013.1257251152629 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 25 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Рисунок 3.11 — Результат виконання етапа наладки верстата

Завдання поступило на етап виконання верстатом, наладка для нового завдання в цей час не відбувається:

```
CREATE state = 0 quantity = 1015 tnext = 1001.414937904849
PROCESS 0 state = 1 quantity = 1032 tnext = 999.8977614135391 failures = 0 queue size = 3
PROCESS 1 state = 0 quantity = 1032 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 1031 tnext = 1000.3882319788863 failures = 0 queue size = 0
Channel 0: Current task - 1011
Interrupted: false  Left time: 0.0  Full time: 0.4904705653472558
DISPOSE quantity = 1010
PROCESS 3 state = 0 quantity = 45 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 44 tnext = 1013.1257251152629 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 25 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Рисунок 3.12 — Результат виконання етапа виконання верстатом

Поломка верстату. Стан перед виникненням поломки:

```
CREATE state = 0 quantity = 579 tnext = 562.3530738318487
PROCESS 0 state = 1 quantity = 587 tnext = 561.8394447769307 failures = 0 queue size = 4
PROCESS 1 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 586 tnext = 562.0815075169484 failures = 0 queue size = 0
Channel 0: Current task - 574
Interrupted: true  Left time: 0.3440671297537392  Full time: 0.6305649860106515
DISPOSE quantity = 573
PROCESS 3 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 23 tnext = 562.0815075169484 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 15 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Рисунок 3.13 — Результат виконання етапа виконання верстатом

Стан після виникненням поломки, усунення поломки верстату:

```
CREATE state = 0 quantity = 579 tnext = 562.3530738318487
PROCESS 0 state = 1 quantity = 587 tnext = 562.0815075169484 failures = 0 queue size = 5
PROCESS 1 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 573
PROCESS 3 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 5 state = 1 quantity = 15 tnext = 566.1226779225458 failures = 0 queue size = 0
```

Рисунок 3.14 — Результат виконання усунення поломки верстату

Як бачимо виконання завдання було перерване поломкою верстату, запам’ятало свій прогрес виконання, та було направлено назад у чергу, очікує на усунення поломки, для повернення на етап виконання і продовження із моменту на якому виконання перервалось. Процес усунення поломки запущено.

За результатами проведеної верифікації бачимо, що модель відпрацьовує належним чином та задовольняє всі умови задачі.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ

4.1 Дослідження моделі

На першому етапі планування визначимо фактори та відгуки для проведення експериментального дослідження. Мета дослідження: визначити, чи модифікація моделі впливає на середній час виконання завдання. Маємо наступні якісні фактори моделі: звичайна модель, модифікована модель. Відгуком моделі будем вважати середній час виконання завдання.

На другому етапі проведемо тактичне планування експерименту. Визначений час моделювання з огляду на кілька пробних прогонів буде становити 100000 годин. Для визначення перехідного періоду зробимо 5 прогонів із визначеним часом моделювання 100000 годин, побудуємо графік залежності досліджуваного відгуку від часу прогону. Побудований графік зображено на рисунку 5.1

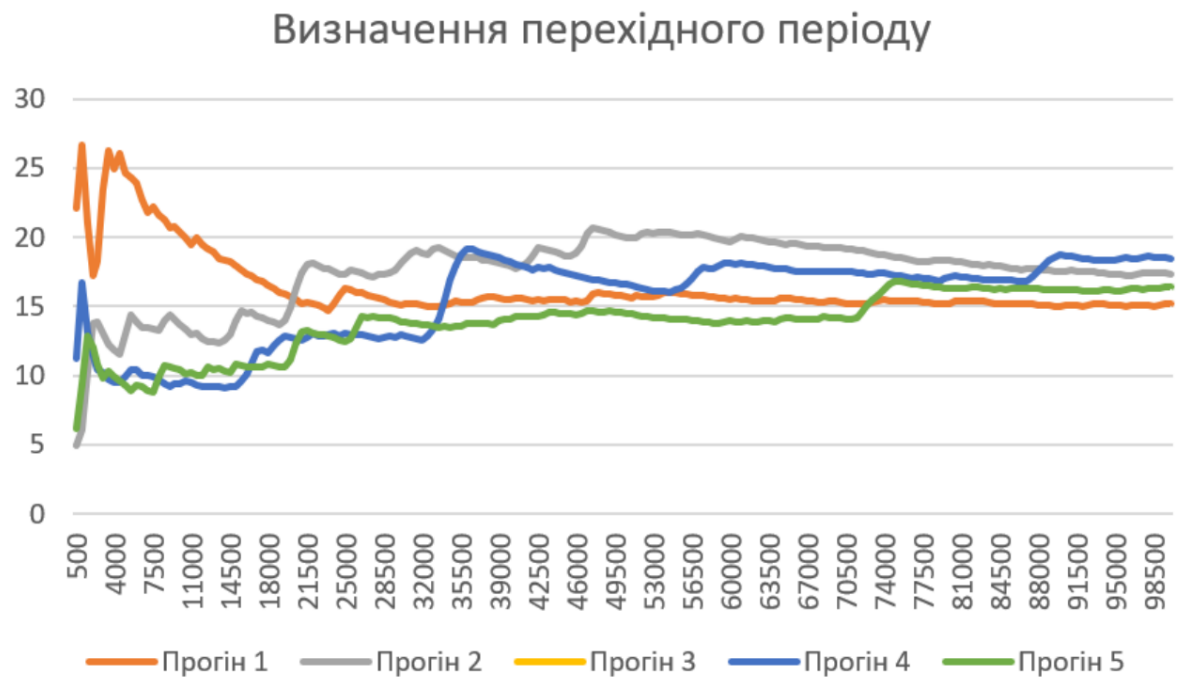


Рисунок 4.1 — Графік залежності досліджуваного відгуку від часу прогону

Маємо результати 5 прогонів на графіку(рис.4.1). Видно, що починаючи із 70000 годин симуляції значення відгуку на всіх прогонах стабілізуються. Зважаючи на отриманий графік залежності досліджуваного відгуку від часу прогону, зазначимо, що тривалість перехідного періоду становить 70000 годин. Отже час моделювання 100000, з яких статистику збиратимемо на проміжку з 70000 до 100000 годин симуляції. Для визначення кількості прогонів використаємо нерівність Чебишева (5.1)

$$p = \frac{\sigma^2}{\varepsilon^2(1-\beta)} \tag{5.1}$$

де, σ^2 – дисперсія відгуку моделі; β – довірча ймовірність; ε – точність вимірювання відгуку моделі.

Середньоквадратичне відхилення часу виконання завдання визначається в результаті симуляції та при тривалості симуляції в 100000 годин дорівнює 16,45. Визначимо β = 0,95 σ = 16,46 , при цьому стандартне відхилення характеризує середню величину відхилення результатів від очікуваного значення. Таким чином, вибір $\varepsilon = \sigma$ = 16,46 дозволяє забезпечити прийнятну точність, відповідну природній варіативності моделі. Звідси визначена кількість прогонів p = 20.

Третім етапом проведемо стратегічне планування експерименту. Проведемо дисперсійний аналіз для визначення впливу модифікації моделі та час виконання завдань. Для кожного з двох можливих варіантів якісних факторів було проведено експеримент із часом моделювання 100000 годин, дані зібрано на проміжку з 70000 до 100000 годин моделювання, кількість прогонів 20, вхідні змінні визначені в умові задачі. Отримані значення середнього часу виконання завданням верстатом з заданого проміжку для всіх прогонів обох експериментів записано у таблицю 4.1.

Таблиця 4.1 — Результати експериментів для дисперсійного аналізу

	Звичайна модель, середній час виконання в годинах	Модифікована модель, середній час виконання в годинах
1	15,83	3,41
2	13,81	3,36
3	14,63	3,35
4	18,04	3,39
5	19,19	3,44
6	13,32	3,36
7	17,89	3,38
8	15,06	3,35
9	14,52	3,38
10	15,77	3,30
11	14,49	3,32
12	15,63	3,36
13	16,74	3,38
14	16,62	3,38
15	15,17	3,37
16	20,54	3,37
17	15,03	3,36
18	18,41	3,31
19	15,37	3,35
20	17,09	3,38
Середнє	16,16	3,37

Звідси маємо середнє значення часу виконання завдання обох експериментів та всіх прогонів(4.1)

$$\bar{y} = \frac{1}{2}\sum_{j=1}^{2} \bar{y_j} = 9{,}765. \quad (4.1)$$

Визначимо значення факторної та залишкової дисперсії:

$$S_{\text{факт}} = \rho \sum_{j=1}^{2} \left(\bar{y_j} - \bar{y}\right)^2 = 1635{,}84; \quad (4.2)$$

$$S = \sum_{i=1}^{p} (y_{i1} - \overline{y_1})^2 + \sum_{i=1}^{p} (y_{i2} - \overline{y_2})^2 = 67{,}43; \quad (4.3)$$

$$d_{\text{факт}} = S_{\text{факт}} = 1635{,}84; \quad (4.4)$$

$$d_{\text{залиш}} = \frac{S_{\text{залиш}}}{q(\rho-1)} = 1{,}7745. \quad (4.5)$$

Визначимо значення критерію Фішера(4.6)

$$F = \frac{d_{\text{факт}}}{d_{\text{залиш}}} = 921{,}85968. \quad (4.6)$$

Критичне значення критерію Фішера визначимо з таблиць за такими параметрами α=0.05, $k_1 = q - 1 = 1$ та $k_2 = q(p-1)$=38 (4.7)

$$F_{\text{кр}} \approx 4{,}1. \quad (4.7)$$

Оскільки F > Fкр , можемо зробити висновок, що модифікація моделі має значний влив на середній час виконання завдання моделлю.

В результаті проведеної роботи можемо зробити низку висновків про звичайну та модифіковану моделі роботи верстату. Для кожної моделі отримано цільові показники визначені умовою задачі, а саме завантаження верстату та статистичні показники часу виконання завдань моделлю.

Основна модифікація передбачала введення додаткової умови: у разі поломки верстата, якщо в черзі знаходиться більше трьох завдань, усі завдання, крім трьох останніх, перенаправляються до субпідрядника. Також враховано можливість переривання ремонту верстата для отримання запасних частин. Причиною цих змін стало бажання оптимізувати продуктивність моделі, скоротити час простою верстата на ремонті та підвищити ефективність виконання завдань.

Під час верифікації було перевірено коректність реалізації всіх етапів процесу: від генерування завдань до обробки, врахування поломок та ремонту. Модель показала стабільні результати в умовах, близьких до реальних. Після проведення верифікації моделі, в ході якої визначено, що модель відпрацьовує

належним чином та задовольняє, всім умовам задачі, в звичайній моделі було відстежено, певну тенденцію до накопичення завдань у черзі через переривання на ремонт верстату, через, що завдання поступали на обробку до верстату із значною затримкою. Модифікована модель була позбавлена цього недоліку за рахунок розвантаження черги, переведенням надлишкових завдань на субпідрядника при поломці верстату. При модифікації також була врахована умова переривання ремонту на пошук додаткових запчастин, але навіть при такій умові модифікація себе проявила значно продуктивнішою, що було детальніше продемонстровано в експериментальному досліджені. Проведено верифікацію модифікованої моделі та перевірено всі умови задачі включно із додатковими, модифікована модель відпрацьовує належним чином.

У ході експериментального дослідження було проведено дисперсійний аналіз для визначення впливу модифікації моделі на середній час виконання завдання. Як фактори розглядалися дві варіації моделі: звичайна та 67 модифікована. Відгуком моделі було обрано середній час виконання завдання. За результатами обчислень, значення статистики Фішера F перевищило критичне значення $F_{\text{кр}}$ = 4,1. Це дозволяє зробити висновок, що модифікація моделі має статистично значущий вплив на середній час виконання завдань. Отримані результати підтверджують доцільність впровадження змін у модель для підвищення її ефективності.

Для покращенні продуктивності системи рекомендується значно знизити швидкість ремонту верстату за рахунок скороченні кількості етапів ремонту, що дозволить значно знизити час виконання завдання, як показали тестові прогони проведені під час верифікацій моделей. Також оптимізувати розподіл ресурсів для ремонту, щоб зменшити час простою.

Для подальшого покращення системи пропонується: дослідити вплив більшої кількості змінних, таких як різна продуктивність субпідрядників чи вартість перенаправлення завдань, впровадити стратегії прогнозування для зменшення ймовірності поломок, наприклад, шляхом аналізу історичних даних.

ВИСНОВКИ

У даній роботі було розроблено і досліджено імітаційну модель функціонування верстата в умовах виробничого процесу з урахуванням ймовірності виникнення поломок. Проведено аналіз задачі, сформульовано вимоги до моделі, обґрунтовано вибір методів моделювання та засобів реалізації. Реалізовано базову та модифіковану версії моделі, що відрізняються наявністю додаткових умов, зокрема перенаправленням завдань до субпідрядника та можливістю переривання ремонту.

Для кожної моделі визначено вхідні та вихідні параметри, використано випадкові величини з відповідними законами розподілу, побудовано структурні схеми. Формалізацію здійснено в рамках теорії масового обслуговування, а програмну реалізацію — з використанням дискретно-подійного підходу.

Коректність реалізації перевірено шляхом верифікації — порівняння результатів симуляції з очікуваними закономірностями. Під адекватністю моделі розуміється її здатність відтворювати логіку роботи реальної системи та коректно реагувати на зміну параметрів.

У ході експериментального дослідження проведено серію запусків моделі з варіюванням параметрів. Результати оброблено за допомогою дисперсійного аналізу, який підтвердив статистично значущий вплив модифікацій на середній час виконання завдань.

За підсумками дослідження встановлено, що модифікована модель забезпечує зменшення часу перебування завдань у системі та підвищує ефективність функціонування. Сформульовано рекомендації щодо подальшого вдосконалення моделі та розширення дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java Documentation – Java Platform, Standard Edition URL: https://docs.oracle.com/en/java/ (дата звернення: 01.05.2025).

2. Arena Simulation Software URL: https://aws.amazon.com/what-is/virtualization/?nc1=h_ls (дата звернення: 16.04.2025).

3. Kelton W. D., Ivey N. and Zupick N. Simulation with Arena, 7th Edition n (дата звернення: 16.04.2025).

4. Wikipedia – Теорія масового обслуговування URL: Теорія масового обслуговування — Вікіпедія (дата звернення: 17.04.2025).

5. Томашевський В. М. Моделювання систем. Київ: Видавнича група ВНУ, 2005. 352 с. (дата звернення: 17.04.2025)

6. Кисельов С. М. Теорія масового обслуговування. Львів: Видавництво Львівської політехніки, 2018. 180 с.

7. Law A. M., Kelton W. D. Simulation Modeling and Analysis, 5th Edition. McGraw-Hill, 2014.

8. Fishman G. S. Discrete-Event Simulation: Modeling, Programming, and Analysis. Springer, 2001.

9. Banks J., Carson J., Nelson B., Nicol D. Discrete-Event System Simulation, 5th Edition. Pearson Education, 2010.

10. Жданов В. П. Системне моделювання: теорія і практика. Харків: ХНУРЕ, 2012.

11. Wikipedia – Ерланговий розподіл. URL: https://uk.wikipedia.org/wiki/Ерланговий_розподіл (дата звернення: 17.04.2025).

12. Wikipedia – Нормальний розподіл. URL: https://uk.wikipedia.org/wiki/Нормальний_розподіл (дата звернення: 17.04.2025).

13. Wikipedia – Експоненціальний розподіл. URL: https://uk.wikipedia.org/wiki/Експоненціальний_розподіл(дата звернення: 17.04.2025).

14. Jain R. The Art of Computer Systems Performance Analysis. Wiley-Interscience, 1991.

15. Trivedi K. S. Probability and Statistics with Reliability, Queuing, and Computer Science Applications. Wiley, 2002.

16. Ross S. M. Introduction to Probability Models, 11th ed. Academic Press, 2014.

17. Шеннон Р. Імітаційне моделювання систем. М.: Мир, 1985.

18. Мищенко В. П. Чисельне моделювання. К.: Либідь, 2011.

19. Java Random API. URL: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html (дата звернення: 01.05.2025).

20. Java Math API. URL: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html(дата звернення: 01.05.2025).

21. Oracle Java Tutorials. URL: https://docs.oracle.com/javase/tutorial/ (дата звернення: 01.05.2025).

22. ISO 9241-210:2010 Ergonomics of human-system interaction – Part 210.

23. ISO 9001:2015 Quality management systems – Requirements.

24. Montgomery D. C. Design and Analysis of Experiments, 9th Edition. Wiley, 2017.

25. Jain A. Queuing Theory and Telecommunications: Networks and Applications. Springer, 2016.

26. Schruben, L. W. Simulation Modeling with Event Graphs. — Management Science, Vol. 29, No. 10, 1983.

27. Discrete Event Simulation – AnyLogic Documentation. URL: https://help.anylogic.com/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fsim%2FDiscrete-event-simulation.html (дата звернення: 10.05.2025).

28. Oracle. Creating Random Numbers Using Java. URL: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html (дата звернення: 10.05.2025).

29. Stack Overflow. How to implement exponential distribution in Java? URL: https://stackoverflow.com/questions/2106503/pseudorandom-number-generator-exponential-distribution (дата звернення: 10.05.2025).

30. Queuing Theory and Modeling — Simul8. URL:https://www.simul8.com/support/help/doku.php?id=features:queuing_theory (дата звернення: 10.05.2025).

КОД РЕАЛІЗАЦІЇ ІМІТАЦІЙНОЇ МОДЕЛІ

Main.java

```
import smo.*;
import smo.Process;
import verstat.*;
import verstatmod.ModelWorkTableMod;
import verstatmod.Process0Mod;
import
verstatmod.Process5SolvingBreakdown;

public class Main {
    public static void main(String[]
args) {
        worktable();
        //worktablemod();
    }

    public static void worktable(){
        CreateVerstatTasks create =
new CreateVerstatTasks("CREATE", 1,
0.0);
        Process0WaitingTask
waitingTask = new
Process0WaitingTask("PROCESS 0", 0);
        Process setupTask = new
Process("PROCESS 1", 0.2, 0.5);
        Process2SolvingTask
solvingTask = new
Process2SolvingTask("PROCESS 2", 0.5,
0.1);
        Dispose dispose = new
Dispose("DISPOSE");



        create.setDistribution("EXP");

waitingTask.setDistribution("NONE");

setupTask.setDistribution("UNIF");

solvingTask.setDistribution("NORM");



        Process waitingBreakdown = new
Process("PROCESS 3", 0);
        Process occurrenceBreakdown =
new Process("PROCESS 4", 20, 2.0);
        Process solvingBreakdown = new
Process("PROCESS 5", 3, 0.75);



waitingBreakdown.setDistribution("NONE
");

occurrenceBreakdown.setDistribution("N
ORM");

solvingBreakdown.setDistribution("ERL"
);




solvingTask.setOccurrenceBreakProcess(
occurrenceBreakdown);

waitingBreakdown.initializeChannelsWit
hTasks(1);



        //маршрут виконання
        create.addRoutes(
                new Route(waitingTask)
        );
        waitingTask.addRoutes(
                new Route(setupTask,
(Task task) -> setupTask.getState() ==
1 || solvingTask.getState() == 1 ||
solvingBreakdown.getState() == 1)
        );
        setupTask.addRoutes(
                new Route(solvingTask)
        );
        solvingTask.addRoutes(
                new Route(dispose, 1,
1),
                new Route(waitingTask,
1, 2, (Task task) ->
solvingBreakdown.getState() == 0)
        );

        //маршрут поломки
        waitingBreakdown.addRoutes(
                new
Route(occurrenceBreakdown, (Task task)
-> solvingTask.getState() == 0)
        );
        occurrenceBreakdown.addRoutes(
                new
Route(solvingBreakdown, 1, 1),
                new
Route(waitingBreakdown, 1, 2, (Task
task) -> setupTask.getState() == 0)
        );
        solvingBreakdown.addRoutes(new
Route(waitingBreakdown));



        ModelWorkTable model = new
ModelWorkTable(
                create, waitingTask,
```

```
setupTask, solvingTask,
                dispose,
waitingBreakdown, occurrenceBreakdown,
solvingBreakdown
        );
        model.simulate(1000);
    }

    public static void worktablemod(){
        CreateVerstatTasks create =
new CreateVerstatTasks("CREATE", 1,
0.0);
        Process0Mod waitingTask = new
Process0Mod("PROCESS 0", 0);
        Dispose disposeSubcontractor =
new Dispose("DISPOSE SUBCONTRACTOR");
        Process setupTask = new
Process("PROCESS 1", 0.2, 0.5);
        Process2SolvingTask
solvingTask = new
Process2SolvingTask("PROCESS 2", 0.5,
0.1);
        Dispose dispose = new
Dispose("DISPOSE");



        create.setDistribution("EXP");

waitingTask.setDistribution("NONE");

setupTask.setDistribution("UNIF");

solvingTask.setDistribution("NORM");

        Process waitingBreakdown = new
Process("PROCESS 3", 0);
        Process occurrenceBreakdown =
new Process("PROCESS 4", 20, 2.0);
        Process5SolvingBreakdown
solvingBreakdown = new
Process5SolvingBreakdown("PROCESS 5",
3, 0.75);



waitingBreakdown.setDistribution("NONE
");

occurrenceBreakdown.setDistribution("N
ORM");

solvingBreakdown.setDistribution("ERL"
);


        Create createInterrupt = new
Create("CREATE INTERRUPTS", 100, 0.0);
        Process search = new
Process("PROCESS 6", 3);
        Dispose happenedInterrupts =
new Dispose("DISPOSE INTERRUPTS");

createInterrupt.setDistribution("EXP")
;

search.setDistribution("NONE");


waitingTask.setDisposeSubcontractor(di
sposeSubcontractor);

waitingTask.setSolvingBreakdown(solvin
gBreakdown);

waitingBreakdown.initializeChannelsWit
hTasks(1);

solvingTask.setOccurrenceBreakProcess(
occurrenceBreakdown);

solvingBreakdown.setSearchProcess(sear
ch);



        //маршрут виконання
        create.addRoutes(
                new Route(waitingTask)
        );
        waitingTask.addRoutes(
                new Route(
                        setupTask, 1,
1,
                        (Task task) ->
setupTask.getState() == 1 ||
solvingTask.getState() == 1 ||
solvingBreakdown.getState() == 1),
                new Route(

disposeSubcontractor, 1, 2,
                        (Task task) ->
!(waitingTask.getQueueSize() > 3 &&
solvingBreakdown.getState() == 1))
        );
        setupTask.addRoutes(
                new Route(solvingTask)
        );
        solvingTask.addRoutes(
                new Route(dispose, 1,
1),
                new Route(waitingTask,
1, 2, (Task task) ->
solvingBreakdown.getState() == 0)
        );


        //маршрут переривання на пошук
запчастин
        createInterrupt.addRoutes(
                new Route(search)
        );
        search.addRoutes(
                new
Route(happenedInterrupts)
        );
```

```
        //маршрут поломки
        waitingBreakdown.addRoutes(
                new
Route(occurrenceBreakdown, (Task task)
-> solvingTask.getState() == 0)
        );
        occurrenceBreakdown.addRoutes(
                new
Route(solvingBreakdown, 1, 1),
                new
Route(waitingBreakdown, 1, 2, (Task
task) -> setupTask.getState() == 0)
        );
        solvingBreakdown.addRoutes(new
Route(waitingBreakdown));




        ModelWorkTableMod model = new
ModelWorkTableMod(
                create,waitingTask,
disposeSubcontractor,  setupTask,
solvingTask,
                dispose,
waitingBreakdown, occurrenceBreakdown,
solvingBreakdown,
                createInterrupt,
search, happenedInterrupts
        );
        model.simulate(1000);
    }
}
```

Create.java

```
package smo;

public class Create extends Element {
    private int failures = 0;

    public Create(String name, double
delay, double initialTNext) {
        super(name, delay);
        super.setTNext(initialTNext);
    }

    protected Task createTask() {
        return new
Task(super.getTCurr());
    }

    @Override
    public void outAct() {
        super.outAct();

super.setTNext(super.getTCurr() +
super.getDelay());
        Task createdTask =
createTask();
        Route nextRoute =
super.getNextRoute(createdTask);
        if (nextRoute.getElement() ==
null ||
nextRoute.isBlocked(createdTask)) {
            failures++;
        } else {

nextRoute.getElement().inAct(createdTa
sk);
        }
    }

    public int getFailures() {return
failures;}

    public void setFailures (int
failures) {this.failures = failures;}
}
```

Dispose.java

```
package smo;

import java.util.ArrayList;

public class Dispose extends Element {
    ArrayList<Task> processedTasks =
new ArrayList<>();

    public Dispose(String name) {
        super(name);
    }

    @Override
    public void inAct(Task task) {
        super.inAct(task);
        processedTasks.add(task);
        super.outAct();
    }

    @Override
    public void printInfo() {
        System.out.println(getName() +
" quantity = " + getQuantity());
    }

    public ArrayList<Task>
getProcessedTasks() {
        return processedTasks;
    }
}
```

Element.java

```
package smo;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class Element {
    private static int nextId = 0;
    private final int id;
    private final String name;

    private final ArrayList<Route>
routes = new ArrayList<>();
    private String routing =
"PRIORITY_CONDITION";
```

```
    private String distribution;
    private double delayMean;
    private double delayDev;

    private double tCurr;
    private double tNext;
    private int state = 0;
    private int quantity = 0;

    public Element(String name) {
        this.name = name;
        id = nextId;
        nextId++;
        tNext = Double.MAX_VALUE;
        tCurr = tNext;
        delayMean = 1.0;
        distribution = "NONE";
    }

    public Element(String name, double
delayMean) {
        this.name = name;
        id = nextId;
        nextId++;
        tNext = 0.0;
        tCurr = tNext;
        this.delayMean = delayMean;
        distribution = "EXP";
    }

    public Element(String name, double
delayMean, double delayDev) {
        this.name = name;
        id = nextId;
        nextId++;
        tNext = 0.0;
        tCurr = tNext;
        this.delayMean = delayMean;
        this.delayDev = delayDev;
        distribution = "NORM";
    }

    public double getDelay() {
        return switch (distribution) {
            case "EXP" ->
FunRand.Exponential(delayMean);
            case "UNIF" ->
FunRand.Uniform(delayMean, delayDev);
            case "NORM" ->
FunRand.Normal(delayMean, delayDev);
            case "ERL" ->
FunRand.Erlang(delayMean, delayDev);
            default -> delayMean;
        };
    }

    public Route getNextRoute(Task
routedTask) {
        if (routes.isEmpty()) {
            return new Route(null);
        }
        return switch (routing) {
            case "PRIORITY_CONDITION"
-> getNextRouteByPriority(routedTask);
            case "PROBABILITY" ->
getNextRouteByProbability(routedTask);
            case "COMB" ->
getNextRouteCombined(routedTask);
            default ->
getNextRouteByPriority(routedTask);
        };
    }

    public void addRoutes(Route...
routes) {

this.routes.addAll(List.of(routes));

this.routes.sort(Comparator.comparingI
nt(Route::getPriority).reversed());
    }

    private static ArrayList<Route>
getUnblockedRoutes(ArrayList<Route>
routes, Task routedTask) {
        ArrayList<Route>
unblockedRoutes = new ArrayList<>();
        for (Route route : routes) {
            if
(!route.isBlocked(routedTask)) {

unblockedRoutes.add(route);
            }
        }
        return unblockedRoutes;
    }

    private Route
getNextRouteByPriority(Task
routedTask) {
        ArrayList<Route>
unblockedRoutes =
getUnblockedRoutes(routes,
routedTask);
        if (unblockedRoutes.isEmpty())
{
            return routes.getFirst();
        }
        return
unblockedRoutes.getFirst();
    }

    private static double[]
getScaledProbabilities(ArrayList<Route
> routes) {
        double[] probabilities = new
double[routes.size()];
        for (int i = 0; i <
routes.size(); i++) {
            probabilities[i] =
routes.get(i).getProbability() + (i ==
0 ? 0 : probabilities[i - 1]);
        }
        for (int i = 0; i <
probabilities.length; i++) {
            probabilities[i] *= 1 /
(probabilities[probabilities.length -
```

```
1]);
        }
        return probabilities;
    }

    private Route
getNextRouteByProbability(Task
routedTask) {
        ArrayList<Route>
unblockedRoutes =
getUnblockedRoutes(routes,
routedTask);
        if (unblockedRoutes.isEmpty())
{
            return routes.getFirst();
        }
        double probability =
Math.random();
        double[] scaledProbabilities =
getScaledProbabilities(unblockedRoutes
);
        for (int i = 0; i <
scaledProbabilities.length; i++) {
            if (probability <
scaledProbabilities[i]) {
                return
unblockedRoutes.get(i);
            }
        }
        return
unblockedRoutes.getLast();
    }

    private ArrayList<Route>
findRoutesByPriority(int priority) {
        ArrayList<Route>
routesByPriority = new ArrayList<>();
        for (Route route : routes) {
            if (route.getPriority() ==
priority) {

routesByPriority.add(route);
            }
        }
        return routesByPriority;
    }

    private Route
getNextRouteCombined(Task routedTask)
{
        Route selectedRoute = null;

        for (Route route : routes) {
            if
(!route.isBlocked(routedTask)) {
                selectedRoute = route;
                break;
            }
        }

        if (selectedRoute == null) {
            return routes.getFirst();
        }

        ArrayList<Route>
samePriorityRoutes =
findRoutesByPriority(selectedRoute.get
Priority());
        double probability =
Math.random();
        double[] scaledProbabilities =
getScaledProbabilities(samePriorityRou
tes);

        for (int i = 0; i <
scaledProbabilities.length; i++) {
            if (probability <
scaledProbabilities[i]) {
                selectedRoute =
samePriorityRoutes.get(i);
                break;
            }
        }

        return selectedRoute;
    }

    public void inAct(Task task) {}

    public void outAct() {quantity++;}

    public double getTNext() {return
tNext;}

    public void setTNext(double tNext)
{this.tNext = tNext;}

    public double getTCurr() {return
tCurr;}

    public void setTCurr(double tCurr)
{this.tCurr = tCurr;}

    public int getId() {return id;}

    public void setRouting(String
routing) {
        this.routing = routing;
    }

    public void setDistribution(String
distribution) {this.distribution =
distribution;}

    public double getDelayMean() {
        return delayMean;
    }

    public void setDelayMean(double
delayMean) {
        this.delayMean = delayMean;
    }

    public double getDelayDev() {
        return delayDev;
    }

    public void setDelayDev(double
```

```
delayDev) {
        this.delayDev = delayDev;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int
quantity) {
        this.quantity = quantity;
    }

    public void changeQuantity(int
delta) {
        this.quantity += delta;
    }

    public int getState() {return
state;}

    public void setState(int state)
{this.state = state;}

    public String getName() {return
name;}

    public void printInfo() {
        System.out.println(name + "
state = " + getState() + " quantity =
" + getQuantity() + " tnext = " +
getTNext());
    }

    public void printResult() {
        System.out.println(name + "
quantity = " + getQuantity());
    }

    public void doStatistics(double
delta) {}

}
```

FunRand.java

```
package smo;

import java.util.Random;

public class FunRand {
    /**
     * Generates a random value
according to an exponential
     * distribution
     *
     * @param delayMean mean value
     * @return a random value
according to an exponential
     * distribution
     */
    public static double
Exponential(double delayMean) {
        double a = 0;
        while (a == 0) {
            a = Math.random();
        }
        a = -delayMean * Math.log(a);
        return a;
    }

    /**
     * Generates a random value
according to a uniform
     * distribution
     *
     * @param min min value
     * @param max max value
     * @return a random value
according to a uniform distribution
     */
    public static double
Uniform(double min, double max) {
        double a = 0;
        while (a == 0) {
            a = Math.random();
        }
        a = min + a * (max - min);
        return a;
    }

    /**
     * Generates a random value
according to a normal (Gauss)
     * distribution
     *
     * @param delayMean      mean
value
     * @param delayDev standard
deviation of normal distribution
     * @return a random value
according to a normal (Gauss)
distribution
     */
    public static double Normal(double
delayMean, double delayDev) {
        double a;
        Random r = new Random();
        a = delayMean + delayDev *
r.nextGaussian();
        return a;
    }

    /**
     * Generates a random value
according to an Erlang distribution
     *
     * @param times n
     * @param delayMean  lambda
     * @return a random value
according to an Erlang distribution
     */
    public static double Erlang(double
times, double delayMean) {
        double product = 1.;
        for (int i = 0; i < (int)
times; ++i) {
            double a = Math.random();
            while (a == 0) {
```

```
                a = Math.random();
            }
            product *= a;
        }
        return -Math.log(product) /
delayMean;
    }
}
```

Model.java

```
package smo;
import java.util.ArrayList;
import java.util.Arrays;

public class Model {
    protected final ArrayList<Element>
elements;
    protected double tCurr;
    protected double tNext;
    protected int nearestEvent;
    protected boolean isFirst = true;

    public Model(Element... elements)
{
        this.elements = new
ArrayList<>(Arrays.asList(elements));
        tNext = 0.0;
        tCurr = tNext;
        nearestEvent = 0;
    }

    public void simulate(double time)
{
        while (tCurr < time) {
            tNext = Double.MAX_VALUE;
            for (Element element :
elements) {
                if ((tCurr <
element.getTNext() || isFirst) &&
element.getTNext() < tNext) {
                    tNext =
element.getTNext();
                    nearestEvent =
element.getId();
                }
            }

            updateBlockedElements();


System.out.println("\nEvent in " +
elements.get(nearestEvent).getName() +
", tNext = " + tNext + ", tCurr = " +
tCurr);

            double delta = tNext -
tCurr;
            doModelStatistics(delta);
            for (Element element :
elements) {

element.doStatistics(delta);
            }
            tCurr = tNext;
            for (Element element :
elements) {

element.setTCurr(tCurr);
            }


elements.get(nearestEvent).outAct();
            for (Element element :
elements) {
                if (element.getTNext()
== tCurr) {
                    element.outAct();
                }
            }

            isFirst = false;

            for (Element element :
elements) {
                element.printInfo();
            }
        }
        printResult();
    }

    protected void
updateBlockedElements() {
        for (Element element :
elements) {
            if (element.getTNext() <=
tCurr) {

element.setTNext(tNext);
            }
        }
    }

    public void printResult() {
        System.out.println("\n--------
-----RESULTS-------------");
        for (Element element :
elements) {
            System.out.print("\n");
            element.printResult();

            if (element instanceof
Process p) {

System.out.println("average workload =
" + p.getWorkTime() / tCurr);

System.out.println("failure
probability = " + p.getFailures() /
(double) (p.getQuantity() +
p.getFailures()) + "\n");

System.out.println("average queue = "
+ p.getMeanQueue() / tCurr);
            }

        }
    }
```

```
    protected void
doModelStatistics(double delta) {}
}
```

Pocess.java

```
package smo;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;

public class Process extends Element {
    protected final Deque<Task> queue
= new ArrayDeque<>();
    protected final ArrayList<Channel>
channels = new ArrayList<>();
    protected int failures = 0;
    protected int maxQueueSize =
Integer.MAX_VALUE;
    protected double meanQueue = 0.0;
    protected double workTime = 0.0;
    protected double totalLeaveTime =
0.0;
    protected double previousLeaveTime
= 0.0;

    protected static class Channel {
        private Task currentTask =
null;
        private double tNext =
Double.MAX_VALUE;

        public int getState() {
            return currentTask == null
? 0 : 1;
        }

        public Task getCurrentTask() {
            return currentTask;
        }

        public void
setCurrentTask(Task currentTask) {
            this.currentTask =
currentTask;
        }

        public double getTNext() {
            return tNext;
        }

        public void setTNext(double
tNext) {
            this.tNext = tNext;
        }
    }

    public Process(String name, double
delayMean) {
        super(name, delayMean);
        channels.add(new Channel());
    }

    public Process(String name, double
delayMean, double delayDev) {
        super(name, delayMean,
delayDev);
        channels.add(new Channel());
    }

    public void
initializeChannelsWithTasks(int
tasksNum) {
        tasksNum = Math.min(tasksNum,
channels.size());
        for (int i = 0; i < tasksNum;
i++) {

channels.get(i).setCurrentTask(new
Task(0.0));

channels.get(i).setTNext(super.getTCur
r() + super.getDelay());
        }
    }

    @Override
    public void inAct(Task task) {
        Channel freeChannel =
getFreeChannel();
        if (freeChannel != null) {

freeChannel.setCurrentTask(task);

freeChannel.setTNext(super.getTCurr()
+ super.getDelay());
        } else {
            if (queue.size() <
getMaxQueueSize()) {
                queue.add(task);
            } else {
                failures++;
            }
        }
    }

    @Override
    public void outAct() {
        processCurrentTasks();
        startNextTasks();
    }

    protected void
processCurrentTasks() {
        ArrayList<Channel>
channelsWithMinTNext =
getChannelsWithMinTNext();
        for (Channel channel :
channelsWithMinTNext) {
            Task task =
channel.getCurrentTask();

            Route nextRoute =
getNextRoute(task);
            if
```

```
(nextRoute.isBlocked(task)) {
                continue;
            }

            if (nextRoute.getElement()
!= null) {

task.setTimeOut(super.getTCurr());

nextRoute.getElement().inAct(task);
            }


channel.setCurrentTask(null);

channel.setTNext(Double.MAX_VALUE);

            changeQuantity(1);
            totalLeaveTime +=
super.getTCurr() - previousLeaveTime;
            previousLeaveTime =
super.getTCurr();
        }
    }

    protected void startNextTasks() {
        Channel freeChannel =
getFreeChannel();
        while (!queue.isEmpty() &&
freeChannel != null) {
            Task task = queue.poll();

freeChannel.setCurrentTask(task);

freeChannel.setTNext(super.getTCurr()
+ super.getDelay());
            freeChannel =
getFreeChannel();
        }
    }

    @Override
    public int getState() {
        int state = 0;
        for (Channel channel :
channels) {
            state |=
channel.getState();
        }
        return state;
    }

    @Override
    public double getTNext() {
        double tNext =
Double.MAX_VALUE;
        for (Channel channel :
channels) {
            if (channel.getTNext() <
tNext) {
                tNext =
channel.getTNext();
            }
        }
        return tNext;
    }

    @Override
    public void setTNext(double tNext)
{
        double previousTNext =
getTNext();
        for (Channel channel :
channels) {
            if (channel.getTNext() ==
previousTNext) {

channel.setTNext(tNext);
            }
        }
    }

    protected ArrayList<Channel>
getChannelsWithMinTNext() {
        ArrayList<Channel>
channelsWithMinTNext = new
ArrayList<>();
        double minTNext =
Double.MAX_VALUE;
        for (Channel channel :
channels) {
            if (channel.getTNext() <
minTNext) {
                minTNext =
channel.getTNext();
            }
        }
        for (Channel channel :
channels) {
            if (channel.getTNext() ==
minTNext) {

channelsWithMinTNext.add(channel);
            }
        }
        return channelsWithMinTNext;
    }

    protected Channel getFreeChannel()
{
        for (Channel channel :
channels) {
            if (channel.getState() ==
0) {
                return channel;
            }
        }
        return null;
    }

    public int getMaxQueueSize() {
        return maxQueueSize;
    }

    public int getFailures() {
        return failures;
    }
```

```
    public double getMeanQueue() {
        return meanQueue;
    }

    public double getWorkTime() {
        return workTime;
    }

    @Override
    public void doStatistics(double
delta) {
        super.doStatistics(delta);
        meanQueue += queue.size() *
delta;
        workTime += getState() *
delta;
    }

    @Override
    public void printInfo() {
        System.out.println(getName() +
                " state = " +
getState() +
                " quantity = " +
getQuantity() +
                " tnext = " +
getTNext() +
                " failures = " +
failures +
                " queue size = " +
queue.size()
        );
    }

    public int getQueueSize() {
        return queue.size();
    }
}
```

Route.java

```
package smo;

public class Route {
    private final Element element;
    private int priority = 0;
    private double probability = 1.0;
    private Block block = null;

    public Route(Element element) {
        this.element = element;
    }

    public Route(Element element,
Block block) {
        this.element = element;
        this.block = block;
    }

    public Route(Element element,
double probability) {
        this.element = element;
        this.probability =
probability;
    }

    public Route(Element element,
double probability, int priority) {
        this.element = element;
        this.probability =
probability;
        this.priority = priority;
    }

    public Route(Element element,
double probability, int priority,
Block block) {
        this.element = element;
        this.probability =
probability;
        this.priority = priority;
        this.block = block;
    }

    public boolean isBlocked(Task
task) {
        if (block == null) {
            return false;
        }
        try {
            return block.call(task);
        } catch (Exception e) {
            throw new
RuntimeException(e);
        }
    }

    @FunctionalInterface
    public interface Block {
        Boolean call(Task task);
    }

    public Element getElement() {
        return element;
    }

    public int getPriority() {
        return priority;
    }

    public double getProbability() {
        return probability;
    }

    public void setBlock(Block block)
{
        this.block = block;
    }

}
```

Task.java

```
package smo;

public class Task {
    private static int nextId = 0;
    private final int id;
    private final double timeIn;
    private double timeOut;
```

```
    public Task(double timeIn) {
        this.timeIn = timeIn;
        this.timeOut = timeIn;
        this.id = nextId;
        nextId++;
    }

    public int getId() {
        return id;
    }

    public double getTimeIn() {
        return timeIn;
    }

    public double getTimeOut() {
        return timeOut;
    }

    public void setTimeOut(double
timeOut) {
        this.timeOut = timeOut;
    }
}
```

CreateVerstatTasks.java

```
package verstat;

import smo.Create;
import smo.Task;


public class CreateVerstatTasks
extends Create {
    public CreateVerstatTasks(String
name, double delay, double
initialTNext) {
        super(name, delay,
initialTNext);
    }

    @Override
    protected Task createTask() {
        return new
VerstatTask(super.getTCurr());
    }
}
```

ModelWorkTable.java

```
package verstat;

import smo.Dispose;
import smo.Element;
import smo.Task;
import smo.Model;
import smo.Process;
import java.util.ArrayList;
import java.util.Objects;

public class ModelWorkTable extends
Model {
    private double totalSetupTime;
    private double
totalProcessingTime;

    public
ModelWorkTable(Element...elements) {
        super(elements);
        this.totalSetupTime = 0;
        this.totalProcessingTime = 0;
    }

    // Inner class to encapsulate task
processing statistics
    private static class
TaskStatistics {
        final double avgTimeInSystem;
        final double minTimeInSystem;
        final double maxTimeInSystem;
        final double
standardDeviation;
        final int totalProcessedTasks;

        TaskStatistics(double
avgTimeInSystem, double
minTimeInSystem,
                       double
maxTimeInSystem, double
standardDeviation,
                       int
totalProcessedTasks) {
            this.avgTimeInSystem =
avgTimeInSystem;
            this.minTimeInSystem =
minTimeInSystem;
            this.maxTimeInSystem =
maxTimeInSystem;
            this.standardDeviation =
standardDeviation;
            this.totalProcessedTasks =
totalProcessedTasks;
        }
    }

    // Method to calculate detailed
task processing statistics
    private TaskStatistics
calculateTaskStatistics() {
        ArrayList<Task> processedTasks
= new ArrayList<>();
        for (Element element :
elements) {
            if (element instanceof
Dispose d &&
Objects.equals(d.getName(),
"DISPOSE")) {

processedTasks.addAll(d.getProcessedTa
sks());
            }
        }

        if (processedTasks.isEmpty())
{
            return new
TaskStatistics(0, 0, 0, 0, 0);
```

```
        }

        double avgTimeInSystem =
processedTasks.stream()
                .mapToDouble(task ->
task.getTimeOut() - task.getTimeIn())
                .average()
                .orElse(0.0);

        double minTimeInSystem =
processedTasks.stream()
                .mapToDouble(task ->
task.getTimeOut() - task.getTimeIn())
                .min()
                .orElse(0.0);

        double maxTimeInSystem =
processedTasks.stream()
                .mapToDouble(task ->
task.getTimeOut() - task.getTimeIn())
                .max()
                .orElse(0.0);

        // Обчислення стандартного
відхилення
        double variance =
processedTasks.stream()
                .mapToDouble(task ->
Math.pow((task.getTimeOut() -
task.getTimeIn()) - avgTimeInSystem,
2))
                .average()
                .orElse(0.0);
        double standardDeviation =
Math.sqrt(variance);

        return new TaskStatistics(
                avgTimeInSystem,
                minTimeInSystem,
                maxTimeInSystem,
                standardDeviation,
                processedTasks.size()
        );
    }

   @Override
    public void simulate(double time)
{
        while (tCurr < time) {
            tNext = Double.MAX_VALUE;
            nearestEvent = -1;

            for (Element element :
elements) {
                if ((tCurr <
element.getTNext() || isFirst) &&
element.getTNext() <= tNext) {
                    if
(element.getTNext() == tNext) {
                        if
(element.getName().equals("PROCESS 4")
&&

elements.get(nearestEvent).getName().e
quals("PROCESS 2")) {

nearestEvent = element.getId();
                        }
                    } else {
                        tNext =
element.getTNext();
                        nearestEvent =
element.getId();
                    }
                }
            }

            updateBlockedElements();


System.out.println("\nEvent in " +
elements.get(nearestEvent).getName() +
", tNext = " + tNext + ", tCurr = " +
tCurr);

            double delta = tNext -
tCurr;
            doModelStatistics(delta);
            for (Element element :
elements) {

element.doStatistics(delta);
            }

            tCurr = tNext;
            for (Element element :
elements) {

element.setTCurr(tCurr);
            }


elements.get(nearestEvent).outAct();
            for (Element element :
elements) {
                if (element.getTNext()
== tCurr) {
                    element.outAct();
                }
            }

            isFirst = false;

            for (Element element :
elements) {
                element.printInfo();
            }
        }
        printResult();
    }


    private double
getAverageTaskTimeInSystem() {
        ArrayList<Task> processedTasks
= new ArrayList<>();
        for (Element element :
elements) {
```

```
            if (element instanceof
Dispose d &&
Objects.equals(d.getName(),
"DISPOSE")) {

processedTasks.addAll(d.getProcessedTa
sks());
            }
        }
        if (processedTasks.isEmpty())
{
            return 0;
        }
        double totalTasksTimeInSystem
= 0;
        for (Task task :
processedTasks) {
            totalTasksTimeInSystem +=
task.getTimeOut() - task.getTimeIn();
        }
        return totalTasksTimeInSystem
/ processedTasks.size();
    }

    // Calculate workload for setup
and solving processes
    private double
workloadSetupSolving(){
        double
settingAndSolvingWorkTime = 0;
        for (Element element :
elements) {
            if (element instanceof
Process p &&
(Objects.equals(element.getName(),
"PROCESS 1") ||
Objects.equals(element.getName(),
"PROCESS 2"))) {

settingAndSolvingWorkTime +=
p.getWorkTime();
            }
        }
        return
(settingAndSolvingWorkTime / tCurr);
    }

    public double
getResolutionBreakDownTime(){
        double result = 0;
        for (Element element :
elements) {
            if (element instanceof
Process p) {
                if
(Objects.equals(p.getName(), "PROCESS
5")) {
                    result =
p.getWorkTime();
                }
            }
        }
        return result;
    }
```

```
    @Override
    public void printResult() {
        System.out.println("\n--------
-----RESULTS-------------");

        // Task statistics
        TaskStatistics taskStats =
calculateTaskStatistics();

        System.out.printf("1)
worktable workload: %.2f%%%n",
workloadSetupSolving() * 100);
        System.out.printf("2) average
task time in system: %.2f hours%n",
getAverageTaskTimeInSystem());

        System.out.printf("task
statistics:%n");
        System.out.printf("   -
minimum time in system: %.2f hours%n",
taskStats.minTimeInSystem);
        System.out.printf("   -
average time in system: %.2f hours%n",
taskStats.avgTimeInSystem);
        System.out.printf("   -
maximum time in system: %.2f hours%n",
taskStats.maxTimeInSystem);
        System.out.printf("   -
standard deviation: %.2f hours%n",
taskStats.standardDeviation);
        System.out.printf("   - total
tasks processed: %d%n",
taskStats.totalProcessedTasks);

        // setup and processing time
        for (Element element :
elements) {
            if (element instanceof
Process p) {
                if
(Objects.equals(p.getName(), "PROCESS
1")) {
                    totalSetupTime =
p.getWorkTime();
                } else if
(Objects.equals(p.getName(), "PROCESS
2")) {

totalProcessingTime = p.getWorkTime();
                }
            }
        }

        System.out.printf("total
solving time:%n");
        System.out.printf("   - total
setup time: %.2f hours%n",
totalSetupTime);
        System.out.printf("   - total
processing time: %.2f hours%n",
totalProcessingTime);
        System.out.printf("   - total
solving breakdown time: %.2f hours%n",
```

```
getResolutionBreakDownTime());
        System.out.printf("   -
waisting time with upset work: %.2f
hours%n", tCurr-
getResolutionBreakDownTime()-
totalProcessingTime-totalSetupTime);
        System.out.printf("   - total
simulation time: %.2f hours%n",
tCurr);
    }
}
```

Process0WaitingTask.java

```
package verstat;
import smo.Process;
import smo.Task;
import java.util.ArrayDeque;
import java.util.Deque;


public class Process0WaitingTask
extends Process {
    public Process0WaitingTask(String
name, double delayMean) {
        super(name, delayMean);
    }

    @Override
    public void inAct(Task task) {
        // When a task returns after
interruption, it should be placed at
the front of the queue
        if (task instanceof
VerstatTask VerstatTask &&
VerstatTask.isInterrupted()) {
            Deque<Task> tempQueue =
new ArrayDeque<>(queue);
            queue.clear();
            queue.add(task);
            queue.addAll(tempQueue);
        } else {
            // Normal queueing for new
tasks if they are not interrupted
            super.inAct(task);
        }
    }
}
```

Process2SolvingTask.java

```
package verstat;

import smo.Task;
import smo.Process;

public class Process2SolvingTask
extends Process {

    private Process
occurrenceBreakProcess = null; //
Процес 4

    public Process2SolvingTask(String
name, double delayMean, double
delayDev) {
        super(name, delayMean,
delayDev);
    }

    public void
setOccurrenceBreakProcess(Process
occurrenceBreakProcess) {
        this.occurrenceBreakProcess =
occurrenceBreakProcess;
    }

    @Override
    public void inAct(Task task) {
        if(task instanceof VerstatTask
VerstatTask){
            if
(VerstatTask.isInterrupted()) {
                Channel freeChannel =
getFreeChannel();
                if (freeChannel !=
null) {

freeChannel.setCurrentTask(VerstatTask
);
                    double tnext =
super.getTCurr() +
VerstatTask.getLeftTime();
                    if(tnext <
occurrenceBreakProcess.getTNext()){

freeChannel.setTNext(tnext);
                    }else{

VerstatTask.setLeftTime(tnext -
occurrenceBreakProcess.getTNext());

freeChannel.setTNext(occurrenceBreakPr
ocess.getTNext()); // + 0.00000001
                    }

                } else if
(queue.size() < getMaxQueueSize()) {

queue.add(VerstatTask);
                } else {
                    failures++;
                }
            } else {
                // Звичайна логіка для
нових завдань
                Channel freeChannel =
getFreeChannel();
                if (freeChannel !=
null) {

freeChannel.setCurrentTask(VerstatTask
);

                    double
fullTimeToFinishTask  =
super.getDelay();

VerstatTask.setFullTime(fullTimeToFini
```

```
shTask);
                    double tnext =
super.getTCurr() +
fullTimeToFinishTask;
                    if(tnext <
occurrenceBreakProcess.getTNext()){

freeChannel.setTNext(tnext);
                    }else{

VerstatTask.markInterrupted();

VerstatTask.setLeftTime(tnext -
occurrenceBreakProcess.getTNext());

freeChannel.setTNext(occurrenceBreakPr
ocess.getTNext());
                    }

                } else {
                    if (queue.size() <
getMaxQueueSize()) {

queue.add(VerstatTask);
                    } else {
                        failures++;
                    }
                }
            }
        }
    }

    @Override
    protected void startNextTasks() {
        Channel freeChannel =
getFreeChannel();
        while (!queue.isEmpty() &&
freeChannel != null) {
            Task task = queue.poll();

            if (task instanceof
VerstatTask VerstatTask) {
                if
(VerstatTask.isInterrupted()) {


freeChannel.setCurrentTask(VerstatTask
);
                    double tnext =
super.getTCurr() +
VerstatTask.getLeftTime();
                    if (tnext <
occurrenceBreakProcess.getTNext()) {

freeChannel.setTNext(tnext);
                    } else {

VerstatTask.setLeftTime(tnext -
occurrenceBreakProcess.getTNext());

freeChannel.setTNext(occurrenceBreakPr
ocess.getTNext());
                    }
                } else {
                    // Звичайна логіка
для нових завдань

freeChannel.setCurrentTask(VerstatTask
);

                    double
fullTimeToFinishTask =
super.getDelay();

VerstatTask.setFullTime(fullTimeToFini
shTask);

                    double tnext =
super.getTCurr() +
fullTimeToFinishTask;
                    if (tnext <
occurrenceBreakProcess.getTNext()) {

freeChannel.setTNext(tnext);
                    } else {

VerstatTask.markInterrupted();

VerstatTask.setLeftTime(tnext -
occurrenceBreakProcess.getTNext());

freeChannel.setTNext(occurrenceBreakPr
ocess.getTNext());
                    }

                }
            }
            freeChannel =
getFreeChannel();
        }
    }


    @Override
    public void printInfo() {
        super.printInfo();
        for (int i = 0; i <
channels.size(); i++) {
            Channel channel =
channels.get(i);
            Task task =
channel.getCurrentTask();
            if (task == null) {

System.out.println("Channel " + i + ":
No current task.");
            } else {

System.out.println("Channel " + i + ":
Current task - " + task.getId());
                if (task instanceof
VerstatTask VerstatTask) {

System.out.println(

"Interrupted: " +
```

```
VerstatTask.isInterrupted() +
                              "  Left
time: " + VerstatTask.getLeftTime() +
                              "  Full
time: " + VerstatTask.getFullTime());
                }
            }
        }
    }
}
```

VerstatTask.java

```
package verstat;
import smo.Task;

public class VerstatTask extends Task
{
    private boolean isInterrupted;
    private double fullTime;
    private double leftTime;

    public VerstatTask(double timeIn)
{
        super(timeIn);
        this.isInterrupted = false;
        this.fullTime = 0.0;
        this.leftTime = 0.0;
    }

    public void markInterrupted() {
        this.isInterrupted = true;
    }

    public boolean isInterrupted() {
        return isInterrupted;
    }

    public double getFullTime() {
        return fullTime;
    }

    public double getLeftTime() {
        return leftTime;
    }

    public void setFullTime(double
fullTime) {
        this.fullTime = fullTime;
    }

    public void setLeftTime(double
leftTime) {
        this.leftTime = leftTime;
    }
}
```

ModelWorkTableMod.java

```
package verstatmod;

import smo.*;
import verstat.ModelWorkTable;
import java.util.ArrayList;
import java.util.Objects;

public class ModelWorkTableMod extends
ModelWorkTable {
    private int subTasks;
    private int totalTasks;

    public ModelWorkTableMod
(Element...elements) {
        super(elements);
    }

    private double
getSubTaskTotalTask() {
        ArrayList<Task> processedTasks
= new ArrayList<>();
        ArrayList<Task>
subcontractorTasks = new
ArrayList<>();

        for (Element element :
elements) {
            if (element instanceof
Dispose d &&

(Objects.equals(d.getName(),
"DISPOSE") ||

Objects.equals(d.getName(), "DISPOSE
SUBCONTRACTOR"))) {

processedTasks.addAll(d.getProcessedTa
sks());
            }

            if (element instanceof
Dispose d &&

Objects.equals(d.getName(), "DISPOSE
SUBCONTRACTOR")) {

subcontractorTasks.addAll(d.getProcess
edTasks());
            }
        }
        subTasks =
subcontractorTasks.size(); totalTasks
= processedTasks.size();
        if (processedTasks.isEmpty())
{
            return 0;
        }
        return (double)
subcontractorTasks.size() /
processedTasks.size();
    }

    public int getInterrupts(){
        int interrupts = 0;
        for (Element element :
elements) {
            if (element instanceof
Process5SolvingBreakdown p) {
                if
(Objects.equals(p.getName(), "PROCESS
```

```
5")) {
                    interrupts =
p.getInterrupts();
                }
            }
        }
        return interrupts;
    }

    public double
getInterruptResolutionTime(){
        double totalSearchTime =
getInterrupts() * 3;
        return totalSearchTime /
getResolutionBreakDownTime();
    }

    @Override
    public void printResult() {
        super.printResult();
        System.out.println("\n--------
-----MOD RESULTS-------------");
        System.out.printf("3) prop of
subcontractor tasks to total tasks:
%.2f%%%n", getSubTaskTotalTask() *
100);
        System.out.printf("4) prop of
interruption search time to resolution
breakdown time: %.2f%%%n",
getInterruptResolutionTime() * 100);

System.out.println("\nsubcontractor
tasks: " + subTasks);
        System.out.println("total
system processed tasks: " +
totalTasks);

System.out.println("interrupts: " +
getInterrupts());
        System.out.println("interrupts
time: " + (getInterrupts() * 3));
    }
}
```

Process0Mod.java

```
package verstatmod;

import smo.Dispose;
import smo.Task;
import verstat.Process0WaitingTask;
import smo.Process;

public class Process0Mod extends
Process0WaitingTask {
    private Process solvingBreakdown =
null;
    private Dispose
disposeSubcontractor = null;

    public Process0Mod(String name,
double delayMean) {
        super(name, delayMean);
    }

    public void
setSolvingBreakdown(Process
solvingBreakdown) {
        this.solvingBreakdown =
solvingBreakdown;
    }

    public void
setDisposeSubcontractor(Dispose
disposeSubcontractor) {
        this.disposeSubcontractor =
disposeSubcontractor;
    }

    public void
removeTasksForSubcontractor(int
maxRemainingTasks, Dispose
subcontractorDispose) {
        while (queue.size() >
maxRemainingTasks) {
            Task task = queue.poll();
            if (task != null) {

subcontractorDispose.inAct(task); //
Передаємо завдання на субпідрядника
            }
        }
    }


    @Override
    public void inAct(Task task) {
    super.inAct(task);
        // Логіка активації очищення
черги
        if
(solvingBreakdown.getState() == 1) {
// Якщо процес 5 активний (поломка)

removeTasksForSubcontractor(3,
disposeSubcontractor); // Тільки
останні 3 завдання залишаються
        }
    }
}
```

Process5SolvingBreakdown.java

```
package verstatmod;

import smo.Task;
import smo.Process;

public class Process5SolvingBreakdown
extends Process {
    private Process searchProcess =
null;
    private int interrupts = 0;

    public
Process5SolvingBreakdown(String name,
double delayMean, double delayDev) {
        super(name, delayMean,
delayDev);
    }
```

```
    public void
setSearchProcess(Process
searchProcess) {
        this.searchProcess =
searchProcess;
    }

    public int getInterrupts() {
        return interrupts;
    }

    @Override
    public void inAct(Task task) {
        Channel freeChannel =
getFreeChannel();
        if (freeChannel != null) {

freeChannel.setCurrentTask(task);
            if
(searchProcess.getState() == 0){

freeChannel.setTNext(super.getTCurr()
+ super.getDelay());
            }else{
                double planned =
super.getTCurr() + super.getDelay();

freeChannel.setTNext(planned + 3);
                interrupts++;

System.out.println("INTERRUPTION IN
BREAKDOWN" + " planned: " + planned +
" fact: " + freeChannel.getTNext());
            }

        } else {
            if (queue.size() <
getMaxQueueSize()) {
                queue.add(task);
            } else {
                failures++;
            }
        }
    }

    @Override
    protected void startNextTasks() {
        Channel freeChannel =
getFreeChannel();
        while (!queue.isEmpty() &&
freeChannel != null) {
            Task task = queue.poll();

freeChannel.setCurrentTask(task);
            if
(searchProcess.getState() == 0){

freeChannel.setTNext(super.getTCurr()
+ super.getDelay());
            }else{
                double planned =
super.getTCurr() + super.getDelay();

freeChannel.setTNext(planned + 3);
                interrupts++;

System.out.println("INTERRUPTION IN
BREAKDOWN" + " planned: " + planned +
" fact: " + freeChannel.getTNext());
            }
            freeChannel =
getFreeChannel();
        }
    }
}
```

Результати верифікації звичайної моделі

	Вхідні змінні										Вихідні змінні				Додаткові вихідні змінні						
Прогін	Затримка CREATE створення завдання, год		Затримка PROCESS 1 наладка верстату , год		Затримка PROCESS 2 обробка завдання, год		Затримка PROCESS 4 виникнення поломки, год		Затримка PROCESS 5 усунення поломки, год		Завантаження верстату	Мінімальний час виконання завдання, год	Середній час виконання завдання, год	Максимальний час виконання завдання, год	Квадрртичне відхилення часу виконання завданн	Сумарний час наладки верстату, год	Сумарний час обробки завдання верстатом, год	Сумарний час ремонту верстату, год	Сумарний час відсутності завдань для верстату, г	Кількість надійдених завдань у систему	Кількість оброблених завдань системою	Кількість завдань у системі при стопі симуляції
№	exp	unif, min	unif, max	norm, mean	norm, dev	norm, mean	norm, dev	elr, n	elr, mean														
1	1	0,2	0,5	0,5	0,1	20	2	3	0,75	0,86	0,59	10,38	31,31	7,6	359,44	503,98	85,87	50,89	999	996	3
2	**0.1**	0,2	0,5	0,5	0,1	20	2	3	0,75	0,87	1	454,4	899,93	262,12	367,83	507,11	125,06	0	9986	1014	8972
3	1	0,2	**0,22**	0,5	0,1	20	2	3	0,75	0,71	0,42	2,82	11,48	2,33	214,04	496,03	106,08	183,91	1001	998	3
4	1	0,2	0,5	**0,2**	0,1	20	2	3	0,75	0,59	0,24	1,99	10,69	1,93	361,42	236,66	85,5	316,63	1026	1023	3
5	1	0,2	0,5	0,5	0,1	**5**	2	3	0,75	0,73	0,6	90,96	205,22	50,06	314,52	412,89	270,21	2,69	1021	821	200
6	1	0,2	0,5	0,5	0,1	20	2	**1**	0,75	0,84	0,49	3,59	19,21	2,61	349,89	492,44	42,62	115,18	987	981	6

Результати верифікації модифікованої моделі

	Вхідні змінні										Вихідні змінні				Додаткові вихідні змінні						
Прогін	Затримка CREATE створення завдання, год		Затримка PROCESS 1 наладка верстату , год		Затримка PROCESS 2 обробка завдання, год		Затримка PROCESS 4 виникнення поломки, год		Затримка PROCESS 5 усунення поломки, год		Завантаження верстату	Мінімальний час виконання завдання, год	Середній час виконання завдання, год	Максимальний час виконання завдання, год	Квадрртичне відхилення часу виконання завдання, го	Сумарний час наладки верстату, год	Сумарний час обробки завдання верстатом, год	Сумарний час ремонту верстату, год	Сумарний час відсутності завдань для верстату, год	Кількість надійдених завдань у систему	Кількість оброблених завдань системою	Кількість завдань у системі при стопі симуляції	Кількість завдань субпідрядника	Відношення завдань субпідрядника до виконаних	Віношення часу переривань до загального ремонту
№	exp	unif, min	unif, max	norm, mean	norm, dev	norm, mean	norm, dev	elr, n	elr, mean																		
1	1	0,2	0,5	0,5	0,1	20	2	3	0,75	0,75	0,55	2,98	16,45	1,83	313,92	441,19	134,66	110,26	1002	876	4	122	0,12	0,04
2	**0.1**	0,2	0,5	0,5	0,1	20	2	3	0,75	0,88	0,71	18,98	77,1	13,63	367,73	516,63	115,69	0	10107	1021	158	8928	0,89	0,05
3	1	0,2	**0,22**	0,5	0,1	20	2	3	0,75	0,66	0,41	2	14,09	1,55	198,8	465,4	130,57	205,7	1022	923	0	99	0,09	0,04
4	1	0,2	0,5	**0,2**	0,1	20	2	3	0,75	0,54	0,24	1,31	11,66	1,23	331,66	210,5	130,41	327,5	1007	934	3	70	0,07	0
5	1	0,2	0,5	0,5	0,1	**5**	2	3	0,75	0,63	0,58	4,08	19,36	2,1	269,07	365,44	345,39	20,16	1016	697	4	315	0,31	0,01
6	1	0,2	0,5	0,5	0,1	20	2	**1**	0,75	0,86	0,54	3,07	9,63	1,8	357,65	508,8	38,99	94,85	1061	1010	2	49	0,04	0

Завдання поступило на етап наладки верстата:

```
CREATE state = 0 quantity = 1015 tnext = 1001.414937904849
PROCESS 0 state = 1 quantity = 1032 tnext = 999.8378461222414 failures = 0 queue size = 3
PROCESS 1 state = 1 quantity = 1031 tnext = 999.8977614135391 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 1031 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 1010
PROCESS 3 state = 0 quantity = 45 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 44 tnext = 1013.1257251152629 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 25 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Завдання поступило на етап виконання верстатом, наладка для нового завдання в цей час не відбувається:

```
CREATE state = 0 quantity = 1015 tnext = 1001.414937904849
PROCESS 0 state = 1 quantity = 1032 tnext = 999.8977614135391 failures = 0 queue size = 3
PROCESS 1 state = 0 quantity = 1032 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 1031 tnext = 1000.3882319788863 failures = 0 queue size = 0
Channel 0: Current task - 1011
Interrupted: false  Left time: 0.0  Full time: 0.4904705653472558
DISPOSE quantity = 1010
PROCESS 3 state = 0 quantity = 45 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 44 tnext = 1013.1257251152629 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 25 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Поломка верстату. Стан перед виникненням поломки:

```
CREATE state = 0 quantity = 579 tnext = 562.3530738318487
PROCESS 0 state = 1 quantity = 587 tnext = 561.8394447769307 failures = 0 queue size = 4
PROCESS 1 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 586 tnext = 562.0815075169484 failures = 0 queue size = 0
Channel 0: Current task - 574
Interrupted: true  Left time: 0.3440671297537392  Full time: 0.6305649860106515
DISPOSE quantity = 573
PROCESS 3 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 23 tnext = 562.0815075169484 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 15 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
```

Стан після виникненням поломки, усунення поломки верстату:

```
CREATE state = 0 quantity = 579 tnext = 562.3530738318487
PROCESS 0 state = 1 quantity = 587 tnext = 562.0815075169484 failures = 0 queue size = 5
PROCESS 1 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 587 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 573
PROCESS 3 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 0 quantity = 24 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 5 state = 1 quantity = 15 tnext = 566.1226779225458 failures = 0 queue size = 0
```

Стан моделі до виникнення поломки:

```
Event in PROCESS 1, tNext = 797.5512218689256, tCurr = 797.1495875742986
CREATE state = 0 quantity = 1625 tnext = 797.7265481169521
PROCESS 0 state = 1 quantity = 864 tnext = 797.5512218689256 failures = 0 queue size = 49
DISPOSE SUBCONTRACTOR quantity = 755
PROCESS 1 state = 0 quantity = 842 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 841 tnext = 797.6908676558056 failures = 0 queue size = 0
Channel 0: Current task - 1589
Interrupted: true  Left time: 0.2528166902945941  Full time: 0.3924624771746419
DISPOSE quantity = 819
PROCESS 3 state = 0 quantity = 35 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 34 tnext = 797.6908676558056 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 22 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
CREATE INTERRUPTS state = 0 quantity = 14 tnext = 806.9715937700695
PROCESS 6 state = 0 quantity = 14 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
DISPOSE INTERRUPTS quantity = 14
```

Стан моделі після виникнення поломки:

```
Event in PROCESS 4, tNext = 797.6908676558056, tCurr = 797.5512218689256
CREATE state = 0 quantity = 1625 tnext = 797.7265481169521
PROCESS 0 state = 1 quantity = 865 tnext = 797.6908676558056 failures = 0 queue size = 3
DISPOSE SUBCONTRACTOR quantity = 802
PROCESS 1 state = 0 quantity = 842 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 842 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 819
PROCESS 3 state = 0 quantity = 35 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 0 quantity = 35 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 5 state = 1 quantity = 22 tnext = 801.2573243367827 failures = 0 queue size = 0
CREATE INTERRUPTS state = 0 quantity = 14 tnext = 806.9715937700695
PROCESS 6 state = 0 quantity = 14 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
DISPOSE INTERRUPTS quantity = 14
```

Стан моделі перед поломкою із перериванням:

```
Event in CREATE, tNext = 669.203047293159, tCurr = 669.1152716999178
CREATE state = 0 quantity = 679 tnext = 671.6760814184153
PROCESS 0 state = 1 quantity = 604 tnext = 669.203047293159 failures = 0 queue size = 2
DISPOSE SUBCONTRACTOR quantity = 96
PROCESS 1 state = 0 quantity = 596 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 1 quantity = 595 tnext = 669.3439805338297 failures = 0 queue size = 0
Channel 0: Current task - 741
Interrupted: true  Left time: 0.2176183979124744  Full time: 0.44632723182436085
DISPOSE quantity = 579
PROCESS 3 state = 0 quantity = 28 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 1 quantity = 27 tnext = 669.3439805338297 failures = 0 queue size = 0
PROCESS 5 state = 0 quantity = 18 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
CREATE INTERRUPTS state = 0 quantity = 67 tnext = 674.3094597450022
PROCESS 6 state = 1 quantity = 65 tnext = 670.5588942526022 failures = 0 queue size = 1
DISPOSE INTERRUPTS quantity = 65
```

Стан після виникненням поломки із перериванням, усунення поломки верстату із часовою затримкою в 3 години:

```
Event in PROCESS 4, tNext = 669.3439805338297, tCurr = 669.203047293159
INTERRUPTION IN BREAKDOWN planned: 672.1942320246801 fact: 675.1942320246801
CREATE state = 0 quantity = 679 tnext = 671.6760814184153
PROCESS 0 state = 1 quantity = 604 tnext = 669.3439805338297 failures = 0 queue size = 3
DISPOSE SUBCONTRACTOR quantity = 96
PROCESS 1 state = 0 quantity = 596 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 2 state = 0 quantity = 596 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
Channel 0: No current task.
DISPOSE quantity = 579
PROCESS 3 state = 0 quantity = 28 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 4 state = 0 quantity = 28 tnext = 1.7976931348623157E308 failures = 0 queue size = 0
PROCESS 5 state = 1 quantity = 18 tnext = 675.1942320246801 failures = 0 queue size = 0
CREATE INTERRUPTS state = 0 quantity = 67 tnext = 674.3094597450022
PROCESS 6 state = 1 quantity = 65 tnext = 670.5588942526022 failures = 0 queue size = 1
DISPOSE INTERRUPTS quantity = 65
```

Додаток Г

КОД РЕАЛІЗАЦІЇ ДО ПРОВЕДЕННЯ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

Модифіковано клас ModelWorkTable.java для збору статистичних показників:

```
package verstat;
import smo.Dispose;
import smo.Element;
import smo.Task;
import smo.Model;
import smo.Process;
import java.util.ArrayList;
import java.util.Objects;
import java.util.List;


public class ModelWorkTable extends Model {
    private double totalSetupTime;
    private double totalProcessingTime;

    // АНАЛІЗ масиви для даних для аналізу
    private final List<Integer> simulationTimes = new ArrayList<>();
    private final List<Double> avgTaskTimes = new ArrayList<>();

    public ModelWorkTable(Element...elements) {
        super(elements);
        this.totalSetupTime = 0;
        this.totalProcessingTime = 0;
    }

    // Inner class to encapsulate task processing statistics
    private static class TaskStatistics {
        final double avgTimeInSystem;
        final double minTimeInSystem;
        final double maxTimeInSystem;
        final double standardDeviation;
        final int totalProcessedTasks;

        TaskStatistics(double avgTimeInSystem, double minTimeInSystem,
                       double maxTimeInSystem, double standardDeviation,
                       int totalProcessedTasks) {
            this.avgTimeInSystem = avgTimeInSystem;
            this.minTimeInSystem = minTimeInSystem;
            this.maxTimeInSystem = maxTimeInSystem;
            this.standardDeviation = standardDeviation;
            this.totalProcessedTasks = totalProcessedTasks;
        }
    }

    // Method to calculate detailed task processing statistics
    private TaskStatistics calculateTaskStatistics() {
        ArrayList<Task> processedTasks = new ArrayList<>();
        for (Element element : elements) {
            if (element instanceof Dispose d && Objects.equals(d.getName(),
"DISPOSE")) {
                processedTasks.addAll(d.getProcessedTasks());
            }
        }
```

```
        if (processedTasks.isEmpty()) {
            return new TaskStatistics(0, 0, 0, 0, 0);
        }

        double avgTimeInSystem = processedTasks.stream()
                .mapToDouble(task -> task.getTimeOut() - task.getTimeIn())
                .average()
                .orElse(0.0);

        double minTimeInSystem = processedTasks.stream()
                .mapToDouble(task -> task.getTimeOut() - task.getTimeIn())
                .min()
                .orElse(0.0);

        double maxTimeInSystem = processedTasks.stream()
                .mapToDouble(task -> task.getTimeOut() - task.getTimeIn())
                .max()
                .orElse(0.0);

        // Обчислення стандартного відхилення
        double variance = processedTasks.stream()
                .mapToDouble(task -> Math.pow((task.getTimeOut() -
task.getTimeIn()) - avgTimeInSystem, 2))
                .average()
                .orElse(0.0);
        double standardDeviation = Math.sqrt(variance);

        return new TaskStatistics(
                avgTimeInSystem,
                minTimeInSystem,
                maxTimeInSystem,
                standardDeviation,
                processedTasks.size()
        );
    }

   @Override
    public void simulate(double time) {
       // АНАЛІЗ
        double lastSavedTime = 0;

        while (tCurr < time) {
            tNext = Double.MAX_VALUE;
            nearestEvent = -1;

            for (Element element : elements) {
                if ((tCurr < element.getTNext() || isFirst) &&
element.getTNext() <= tNext) {
                    if (element.getTNext() == tNext) {
                        if (element.getName().equals("PROCESS 4") &&

elements.get(nearestEvent).getName().equals("PROCESS 2")) {
                            nearestEvent = element.getId();
                        }
                    } else {
                        tNext = element.getTNext();
                        nearestEvent = element.getId();
                    }
                }
            }
```

```
            updateBlockedElements();

            System.out.println("\nEvent in " +
elements.get(nearestEvent).getName() + ", tNext = " + tNext + ", tCurr = " +
tCurr);

            double delta = tNext - tCurr;
            doModelStatistics(delta);
            for (Element element : elements) {
                element.doStatistics(delta);
            }

            tCurr = tNext;
            for (Element element : elements) {
                element.setTCurr(tCurr);
            }

            elements.get(nearestEvent).outAct();
            for (Element element : elements) {
                if (element.getTNext() == tCurr) {
                    element.outAct();
                }
            }

            isFirst = false;
```

```
            // АНАЛІЗ Збереження даних кожні 500 годин
            if ((int)tCurr>69999 && ((int)tCurr / 500 > (int)lastSavedTime /
500)) {
                lastSavedTime = tCurr;
                simulationTimes.add((int)tCurr);
                avgTaskTimes.add(getAverageTaskTimeInSystem());
            }

            for (Element element : elements) {
                element.printInfo();
            }
        }
        printResult();
        printStatisticsTable();
    }

    private double getAvg() {
        // Перевірка, чи списки не порожні та мають однаковий розмір
        if (simulationTimes.isEmpty() || avgTaskTimes.isEmpty() ||
simulationTimes.size() != avgTaskTimes.size()) {
            throw new IllegalStateException("no data or incorrec.");
        }

        double sum = 0;
        int count = 0;

        for (int i = 0; i < simulationTimes.size(); i++) {
            int time = simulationTimes.get(i);
            if (time >= 70000 && time <= 100000) {
                sum += avgTaskTimes.get(i);
                count++;
            }
        }
```

```
        if (count == 0) {
            throw new IllegalStateException("No data (70000-100000 hours).");
        }

        return sum / count;
    }

    // АНАЛІЗ вивести дані для аналізу
    private void printStatisticsTable() {
        System.out.println("\n--- Simulation Time vs Average Task Time in System
---");
        System.out.printf("%-15s %-20s%n", "Simulation Time", "Average Task
Time");
        for (int i = 0; i < simulationTimes.size(); i++) {
            System.out.printf("%-15d %-20.2f%n", simulationTimes.get(i),
avgTaskTimes.get(i));
        }

        double avg70100 = getAvg();
        System.out.printf("%-20.2f%n", avg70100);
    }

    private double getAverageTaskTimeInSystem() {
        ArrayList<Task> processedTasks = new ArrayList<>();
        for (Element element : elements) {
            if (element instanceof Dispose d && Objects.equals(d.getName(),
"DISPOSE")) {
                processedTasks.addAll(d.getProcessedTasks());
            }
        }
        if (processedTasks.isEmpty()) {
            return 0;
        }
        double totalTasksTimeInSystem = 0;
        for (Task task : processedTasks) {
            totalTasksTimeInSystem += task.getTimeOut() - task.getTimeIn();
        }
        return totalTasksTimeInSystem / processedTasks.size();
    }

    // Calculate workload for setup and solving processes
    private double workloadSetupSolving(){
        double settingAndSolvingWorkTime = 0;
        for (Element element : elements) {
            if (element instanceof Process p &&
(Objects.equals(element.getName(), "PROCESS 1") ||
Objects.equals(element.getName(), "PROCESS 2"))) {
                settingAndSolvingWorkTime += p.getWorkTime();
            }
        }
        return (settingAndSolvingWorkTime / tCurr);
    }

    public double getResolutionBreakDownTime(){
        double result = 0;
        for (Element element : elements) {
            if (element instanceof Process p) {
```

```
                if (Objects.equals(p.getName(), "PROCESS 5")) {
                    result = p.getWorkTime();
                }
            }
        }
        return result;
    }

    @Override
    public void printResult() {
        System.out.println("\n-------------RESULTS-------------");

        // Task statistics
        TaskStatistics taskStats = calculateTaskStatistics();

        System.out.printf("1) worktable workload: %.2f%%%n",
workloadSetupSolving() * 100);
        System.out.printf("2) average task time in system: %.2f hours%n",
getAverageTaskTimeInSystem());

        System.out.printf("task statistics:%n");
        System.out.printf("   - minimum time in system: %.2f hours%n",
taskStats.minTimeInSystem);
        System.out.printf("   - average time in system: %.2f hours%n",
taskStats.avgTimeInSystem);
        System.out.printf("   - maximum time in system: %.2f hours%n",
taskStats.maxTimeInSystem);
        System.out.printf("   - standard deviation: %.2f hours%n",
taskStats.standardDeviation);
        System.out.printf("   - total tasks processed: %d%n",
taskStats.totalProcessedTasks);
taskStats.totalProcessedTasks);

        // setup and processing time
        for (Element element : elements) {
            if (element instanceof Process p) {
                if (Objects.equals(p.getName(), "PROCESS 1")) {
                    totalSetupTime = p.getWorkTime();
                } else if (Objects.equals(p.getName(), "PROCESS 2")) {
                    totalProcessingTime = p.getWorkTime();
                }
            }
        }

        System.out.printf("total solving time:%n");
        System.out.printf("   - total setup time: %.2f hours%n",
totalSetupTime);
        System.out.printf("   - total processing time: %.2f hours%n",
totalProcessingTime);
        System.out.printf("   - total solving breakdown time: %.2f hours%n",
getResolutionBreakDownTime());
        System.out.printf("   - waisting time with upset work: %.2f hours%n",
tCurr-getResolutionBreakDownTime()-totalProcessingTime-totalSetupTime);
        System.out.printf("   - total simulation time: %.2f hours%n", tCurr);
    }
}
```

Додаток Д

Демонстраційні матеріали
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯОСВІТНЬОГОСТУПЕНЯБАКАЛАВРА
на тему: «Імітаційна модель роботи верстатного процесу систем масового обслуговування»

Виконав студент групи КІД-41
Ал Хадам Семі Резгович

Керівник кваліфікаційної роботи:
Бученко Ігор Анатолійович, старший викладач кафедри КІ

Київ – 2025

Мета роботи – Дослідити функціонування верстата в умовах випадкових збоїв.

Об`єкт дослідження – Процес обробки замовлень у автоматизованій системі.

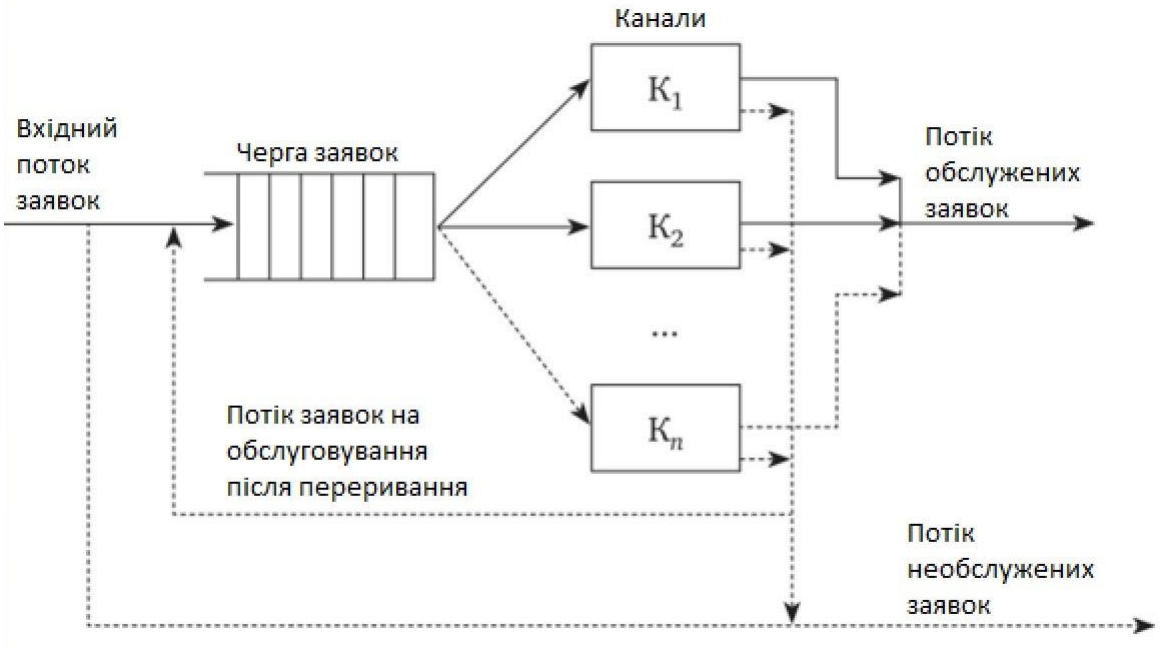
Предмет дослідження – Імітаційна модель роботи верстатного процесу СМО.

Актуальність обраної теми

У наш час ефективність виробничих процесів є ключовим чинником стабільної роботи підприємств. Один із найважливіших елементів таких процесів — верстатне обладнання, яке виконує технологічну обробку продукції. У реальних умовах робота верстатів супроводжується випадковими подіями: поломками, переналагодженням, зміною черги замовлень тощо. Такі події можуть призводити до простоїв, зниження продуктивності та втрат прибутку. У зв'язку з цим особливої актуальності набуває створення імітаційних моделей, які дозволяють заздалегідь дослідити поведінку системи обслуговування, змоделювати різні сценарії навантаження та обрати оптимальні стратегії управління. Завдяки таким моделям можна оцінити завантаження верстата, передбачити критичні ситуації та мінімізувати вплив несподіваних збоїв на виробничий процес.

3

Приклад системи масового обслуговування



4

Характеристики та вимоги до імітаційної моделі

1. реалістичність;
2. стохастичність;
3. гнучкість;
4. масштабованість;
5. облік реальних обмежень;
6. висока точність.

5

FIFO

Принцип "першим прийшов — першим обслуговани"
Елемент, що потрапив до черги першим, обробляється першим.



6

Рисунок – Зображення функціонування концетуальної системи



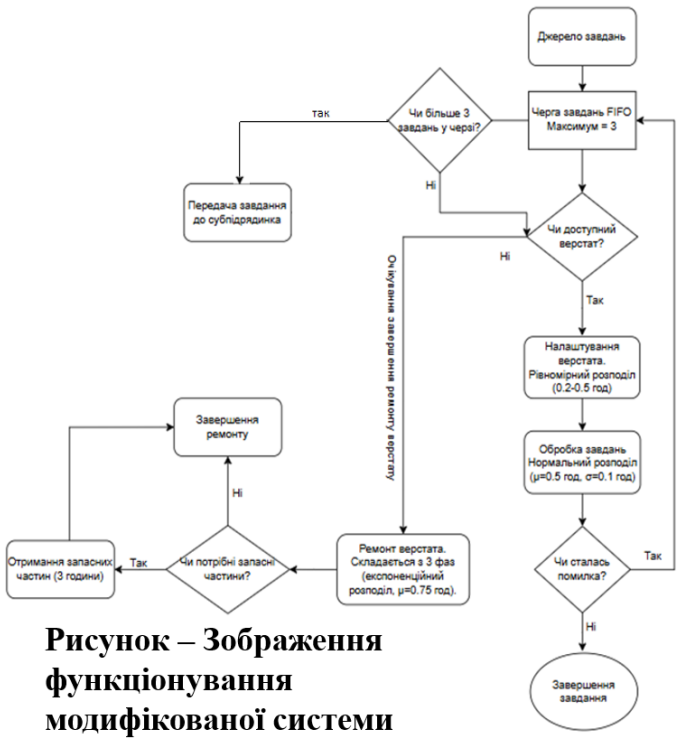
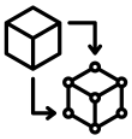


Рисунок – Зображення функціонування модифікованої системи

Етапи розробки та впровадження проєкту

1. Збір даних. Аналіз реальних сценаріїв роботи верстатів, поломок, часу обробки, черг.

2. Розробка концептуальної моделі. Побудова базової логічної схеми роботи СМО.



3. Створення формалізованої моделі. Опис процесів у вигляді взаємодіючих СМО з математичними розподілами.



4. Реалізація імітаційної моделі. Програмування моделі на мові Java з використанням подійного моделювання.



5. Модифікація моделі. Додавання додаткових умов

6. Експериментальні дослідження. Проведення серії симуляцій з різними параметрами.

7. Аналіз результатів . Побудова графіків, порівняння продуктивності моделей, висновки щодо доцільності змін.

ВИСНОВКИ

Дослідивши підходи до побудови імітаційної моделі роботи верстатного процесу в умовах систем масового обслуговування, можна зазначити, що такі моделі є надзвичайно актуальними для сучасного виробництва. Вони дозволяють заздалегідь аналізувати поведінку обладнання в умовах стохастичних подій, таких як поломки, налаштування або перевантаження черги. Особливо важливою є здатність моделі відтворювати нестандартні ситуації, які трапляються на виробництві, та оцінювати вплив змін на продуктивність системи.

Розроблена в ході дослідження імітаційна модель забезпечує врахування реалістичних факторів, таких як можливість переривання ремонту через відсутність запасних частин чи перенаправлення задач до субпідрядника при перевантаженні.

ПРОДОВЖЕННЯ ВИСНОВКІВ

Програмна реалізація моделі на мові Java із застосуванням подійного моделювання дозволила досягти високої гнучкості та точності результатів. Проведені експериментальні симуляції підтвердили ефективність запропонованих модифікацій — зменшення середнього часу перебування завдань у системі та підвищення рівня завантаження верстата без надмірних простоїв.

Інвестиції у розробку імітаційних моделей, подальше вдосконалення алгоритмів обробки подій, а також інтеграція таких моделей у виробничі інформаційні системи є перспективним напрямом, що дозволяє підприємствам приймати обґрунтовані рішення, підвищувати ефективність процесів та забезпечувати стійкість до непередбачуваних ситуацій.