НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

До захисту допущено:
Завідувач кафедри
_____ Наталія СТЕЛЬМАХ
«12» 06 2026 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерно-інтегровані системи та технології в приладобудуванні»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Автоматизована система керування транспортними потоками деталей»

Виконала:
студентка IV курсу, групи ПБ-21
Хохич Аліна Василівна

Керівник:
Доцент, к.т.н., доцент,
Вислоух Сергій Петрович

Рецензент:
Доц. к.н.т., доц. каф. АСНК
Писарець Анна Валеріївна

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2026 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет робототехніки та приладобудування

Кафедра комп'ютерно-інтегрованих технологій виробництва приладів

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Наталія СТЕЛЬМАХ

«12» 06 2021 р.

ЗАВДАННЯ

на дипломну роботу студентці

Хохич Аліні Василівні

1. Тема роботи «**Автоматизована система керування транспортними потоками деталей**», керівник роботи Вислоух Сергій Петрович, к.т.н., доцент, затверджені наказом по університету від «26» 05 2026 р. № 1905-с

2. Термін подання студенткою роботи 07 червня 2026 року

3. Вихідні дані до роботи Задачі керування транспортними потоками деталей при автоматизованому виробництві

4. Зміст роботи: Реферат. Вступ. Огляд стану транспортних потоків деталей а автоматизованому виробництві. Аналіз автоматизованих систем керування транспортними потоками деталей. Основні принципи створення автоматизованих систем керування чергою. Аналіз традиційних методів керування чергою; Вимоги до автоматизованої системи керування чергою з урахуванням сучасних тенденцій. Постановка задачі дипломної роботи, визначення цільової функції та обмежень. Структурно-функціональна схема автоматизованої системи керування. Математична модель багаторівневої системи керування з використанням пріоритетних черг. Алгоритм керування з врахуванням пріоритетних черг. Схема програмного

забезпечення. Інтерфейс користувача при роботі з системою. Інформаційне забезпечення автоматизованої системи. Висновки. Перелік використаних джерел.

5. Перелік ілюстративного матеріалу: Аналіз автоматизованих систем керування транспортними потоками деталей (А1). Структурно-функціональна схема автоматизованої системи (АІ). Математичне забезпечення системи (АІ). Алгоритм керування з врахуванням пріоритетних черг (А1). Структура програмного забезпечення автоматизованої системи (АІ Інтерфейс користувача при роботі з системою (АІ). Інформаційне забезпечення автоматизованої системи з кресленням деталі (А І).

6. Дата видачі завдання 03 березня 2026 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз автоматизованих систем керування транспортними потоками деталей	24.04.2026	
2	Постановка задачі дипломної роботи, визначення цільової функції та обмежень	02.05.2026	
3	Структурно-функціональна схема автоматизованої системи керування.	08.05.2026	
4	Математична модель багаторівневої системи керування з використанням пріоритетних черг	12.05.2026	
5	Алгоритм керування з врахуванням пріоритетних черг	14.05.2026	
6	Програмне забезпечення системи	19.05.2026	
7	Інтерфейс користувача при роботі з системою	22.05.2026	
8	Інформаційне забезпечення автоматизованої системи	29.05.2026	
9	Оформлення дипломної роботи	05.06.2026	

Студентка Аліна ХОХИЧ

Керівник Сергій ВИСЛОУХ

Пояснювальна записка

до дипломної роботи

на тему: **«Автоматизована система керування транспортними потоками деталей»**

Київ – 2026 р.

АНОТАЦІЯ

Дипломна робота на тему: «Автоматизована система керування транспортними потоками деталей» присвячена розробленню системи автоматизованого розподілу комплектуючих між складальними дільницями на основі механізму динамічних пріоритетів. Робота містить графічні матеріали, програмне забезпечення та результати моделювання виробничого процесу.

У першому розділі проведено аналіз існуючих автоматизованих транспортних систем, методів керування матеріальними потоками та підходів до оптимізації виробничої логістики. Розглянуто можливості застосування теорії масового обслуговування та пріоритетних черг для розв'язання задач розподілу деталей між складальними дільницями.

У другому розділі розроблено структурну схему автоматизованої системи, сформовано математичну модель керування потоками деталей та алгоритм багаторівневого визначення пріоритетів.

У третьому розділі описано програмну реалізацію системи, структуру бази даних, інформаційне забезпечення та графічний інтерфейс користувача. Реалізовано моделювання виробничого процесу, контроль якості деталей, механізм перенаправлення комплектуючих та ведення журналу виробничих подій.

У даній дипломній роботі було розроблено автоматизовану систему керування потоками деталей. Робота містить: 127 сторінок, 19 ілюстрації, 6 таблиць, 9 формул, 1 креслень, 2 додатки, 23 літературних джерел.

Ключові слова: автоматизована система керування, матеріальний потік, складальна дільниця, пріоритетна черга, виробнича логістика, моделювання виробничого процесу, база даних.

ABSTRACT

The thesis entitled ″Automated system for managing the flow of parts″ is devoted to the development of an automated system for distributing components between assembly sections based on a dynamic priority mechanism. The work includes graphical materials, software implementation, and production process simulation results.

The first chapter analyzes existing automated transport systems, material flow management methods, and approaches to manufacturing logistics optimization. The applicability of queueing theory and priority queues to the problem of part distribution between assembly sections is considered.

The second chapter presents the structural design of the automated system, the mathematical model of parts flow management, and a multi-level priority assignment algorithm.

The third chapter describes the software implementation of the system, database structure, information support, and graphical user interface. Production process simulation, quality control procedures, component redirection mechanisms, and production event logging have been implemented.

As a result of this thesis, an automated parts flow management system. The thesis contains: 127 pages, 19 illustrations, 6 table, 9 formulas, 1 drawings, 2 appendices, and 23 references.

Keywords: automated control system, material flow, assembly section, priority queue, manufacturing logistics, production process simulation, database.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AGV/ AMR – Automated Guided Vehicle (автоматизований транспортний візок)

AS/RS – Автоматизована складська система

NP – Пріорітетна задача

FIFO – “First In, First Out” — «першим прийшов — першим пішов»

БД – База даних.

ГВС – Гнучка виробнича система

ТМО – Теорія масового обслуговування

RFID – Radio Frequency Identification (радіочастотна ідентифікація)

FPV – First Person View (керування від першої особи)

FC – Flight Controller (польотний контролер)

ESC – Electronic Speed Controller (електронний регулятор швидкості)

ELRS – ExpressLRS (протокол радіокерування)

XT60 – Стандартний силовий електричний роз'єм

ESR – Equivalent Series Resistance (еквівалентний послідовний опір)

GUI – Graphical User Interface (графічний інтерфейс користувача)

API – Application Programming Interface (програмний інтерфейс застосунку)

SQL – Structured Query Language (мова структурованих запитів)

ВСТУП

Актуальність дослідження. Сучасне виробництво потребує ефективних засобів керування матеріальними потоками, оскільки швидкість і точність доставки деталей безпосередньо впливають на продуктивність складальних дільниць. Особливої актуальності це набуває в умовах одночасного виготовлення декількох модифікацій виробів, які можуть використовувати як спільні, так і спеціалізовані комплектуючі.

Традиційні системи транспортування деталей часто використовують фіксовані правила розподілу ресурсів, що не враховують поточний стан виробництва та рівень забезпеченості складальних дільниць. Це може призводити до виникнення дефіциту комплектуючих, простоїв обладнання та зниження ефективності виробничого процесу. Тому актуальним є розроблення автоматизованої системи керування потоками деталей із використанням механізмів пріоритетного розподілу ресурсів.

Метою дипломної роботи є розроблення автоматизованої системи керування потоками деталей між складальними дільницями, яка забезпечує пріоритетний розподіл комплектуючих відповідно до поточних потреб виробництва та дослідження ефективності її роботи шляхом програмного моделювання.

Для досягнення мети дослідження необхідно виконати такі теоретичні й практичні завдання:

– провести аналіз існуючих методів керування матеріальними потоками в автоматизованих виробничих системах;

– дослідити можливості застосування теорії масового обслуговування та пріоритетних черг для розподілу деталей;

– розробити структуру автоматизованої системи керування потоками деталей;

– створити алгоритм визначення пріоритетів складальних дільниць та розподілу комплектуючих;

– розробити інформаційне забезпечення та структуру бази даних системи;

– реалізувати програмне забезпечення для моделювання виробничого процесу;

– виконати дослідження роботи системи в різних виробничих умовах та проаналізувати отримані результати.

РОЗДІЛ 1.
АНАЛІЗ АВТОМАТИЗОВАНИХ СИСТЕМ КЕРУВАННЯ ТРАНСПОРТНИМИ ПОТОКАМИ ДЕТАЛЕЙ

1.1. Роль та значення логістичних процесів у сучасних автоматизованих виробничих системах

Коли ми говоримо про сучасне виробництво, особливо в епоху Четвертої промислової революції (Індустрії 4.0), то зазвичай першими в голову приходять складні роботи, автоматизовані верстати з програмним керуванням чи розумні конвеєри. Проте сама по собі наявність швидкого та дорогого верстата не зробить завод ефективним, якщо заготовки до нього доставляються із запізненням, а готові деталі годинами чекають на підлозі цеху, поки звільниться складське місце. Саме тому внутрішня логістика та керування потоками деталей стали тією основою, яка зв’язує всі технологічні процеси в один працюючий механізм.

Логістика у виробництві — це не просто перевезення вантажів з однієї точки в іншу. Це ціла система планування, координації та безперервного контролю руху матеріалів та інформації. Профільні фахівці з логістичного менеджменту дають чітке і комплексне визначення цьому процесу, яке чудово пояснює його суть:

Логістика – це процес планування, управління та контролю ефективного (за фактором зниження витрат) потоків запасів сировини, матеріалів, незавершеного виробництва, готової продукції, послуг та супутньої інформації від місця виникнення цього потоку до місця його споживання (включаючи експорт, імпорт, внутрішні та зовнішні переміщення) для повного задоволення запитів споживачів [1].

Якщо проаналізувати це визначення ближче до умов конкретного цеху, стає зрозуміло, що рух деталі по технологічному маршруту (наприклад, від складу заготовок через токарні та фрезерні верстати до мийки, посту контролю та складу готової продукції) є класичним логістичним потоком незавершеного виробництва. Головна проблема тут полягає в тому, що у звичайному виробничому циклі деталь проводить безпосередньо на самому верстаті під час обробки лише малу частину часу. Весь інший час — це втрати на очікування, сортування та транспортування.

У наукових дослідженнях, присвячених оптимізації транспортних систем, наводиться важлива статистика щодо структури витрат часу:

Основними частинами виробничого процесу є підготовка виробництва, технологічний процес виготовлення виробу, його транспортування, зберігання і ремонтування. Залежно від самого виробу, від 10 % до 20 % всього виробничого процесу припадає на транспортування [2].

Якщо до цих 10–20% чистого транспортування додати час, який деталі проводять у чергах на проміжних накопичувачах або на складі в очікуванні відбору, виявиться, що логістичні операції займають більшу частину тривалості всього виробничого циклу. Отже, якщо підприємство прагне випускати продукцію швидше й зменшувати її собівартість, потрібно оптимізовувати не лише швидкість роботи інструменту на верстатах, а саме логістичні процеси всередині цехів.

У сучасних гнучких автоматизованих системах роль логістики суттєво змінюється. Раніше, при традиційному підході, кожна ділянка заводу функціонувала автономно: склад намагався просто забити стелажі, транспортний цех возив деталі тоді, коли звільнялися робітники, а виробничі бригади робили норму без прив’язки до реальних потреб наступних ділянок. Це призводило до хаосу, затоварення цехів та постійних затримок.

У спеціалізованій літературі детально описується цей негативний ефект традиційного керування:

При традиційному підході до управління кожна ланка логістичного ланцюжка має власні механізми управління, орієнтовані на власні цілі та критерії ефективності... Параметри результуючого потоку формуються в умовах незалежних управлінських впливів, що здійснюються послідовно в ланках логістичної мережі. Тому з погляду загальних цілей управління параметри результуючого потоку є спонтанними [1].

Автоматизація якраз і покликана прибрати цю спонтанність. Замість ізольованих рішень створюється наскрізне керування. Об'єктом контролю стає весь потік деталей загалом — від першої заготовки до відвантаження. Проте на

практиці виникає серйозний технологічний виклик: автоматизований склад та автоматизований транспорт мають працювати як одне ціле в реальному часі.

Якщо подивитися на складську логістику автоматизованого виробництва, то головне завдання тут — максимально швидко видати потрібний комплект деталей на транспортний засіб (наприклад, конвеєр чи мобільний візок). Але класичні підходи до організації складів часто дають збій, про що свідчать профільні дослідження:

Проблема ефективного розміщення деталей на автоматизованому складі є ключовою для підвищення продуктивності логістичних процесів і дотримання сучасних вимог до швидкості виконання замовлень у межах індустрії 4.0. Класичні підходи до складування − зокрема ABC-аналіз і одноцільова оптимізація за критерієм популярності деталей − часто призводять до нерівномірного розподілу найзатребуваніших позицій між зонами, надмірного навантаження окремих ділянок і необґрунтованого підйому важких компонентів на високі яруси [3].

Коли на складі виникає таке надмірне навантаження окремих ділянок, уся логістична система цеху починає гальмувати. Транспортні засоби змушені стояти в чергах перед зоною видачі, верстати починають простоювати без заготовок, а загальна продуктивність гнучкої лінії падає. Це підтверджує, що автоматизація складської логістики та автоматизація транспортування деталей є абсолютно взаємозалежними процесами. Не можна побудувати розумну транспортну систему цеху, ігноруючи те, як організовані потоки всередині автоматизованого складу.

Таким чином, значення логістичних процесів у сучасних автоматизованих виробничих системах зводиться до кількох **ключових функцій**:

1. **Інтеграційна функція.** Логістика об'єднує роботу складів, транспортних пристроїв та технологічних верстатів у єдиний синхронний ланцюжок.

2. **Синхронізація в реальному часі.** Завдяки автоматизованому обміну даними між системами керування досягається точність передачі деталей без участі людини та виключається людський фактор.

3. **Забезпечення гнучкості.** Сучасні автоматизовані виробництва часто працюють в умовах середньосерійного чи дрібносерійного випуску, де номенклатура виробів постійно змінюється. Саме логістична система дозволяє швидко адаптуватися до нових завдань без повної зупинки ліній за рахунок маневреності транспорту чи динамічного перерозподілу складських комірок.

Раціональна організація логістики сьогодні будується на суворих принципах: єдиноспрямованості (щоб потоки деталей не перетиналися і не створювали заторів), гнучкості, синхронності та загальної оптимізації процесів. Без впровадження цих принципів будь-яка спроба автоматизувати заводи призведе лише до того, що підприємство отримає автоматизований хаос замість реального приросту продуктивності.

Оскільки ми зараз проводимо аналіз того, як транспортні системи функціонують у більшості стандартних випадків (існуючий стан до нашої оптимізації), нижче наведено логіку побудови схеми на рисунку 1.1. яка відображає послідовний, лінійний процес роботи, де рух кожної деталі повністю прив'язаний до жорстких команд від центрального складу або диспетчера.

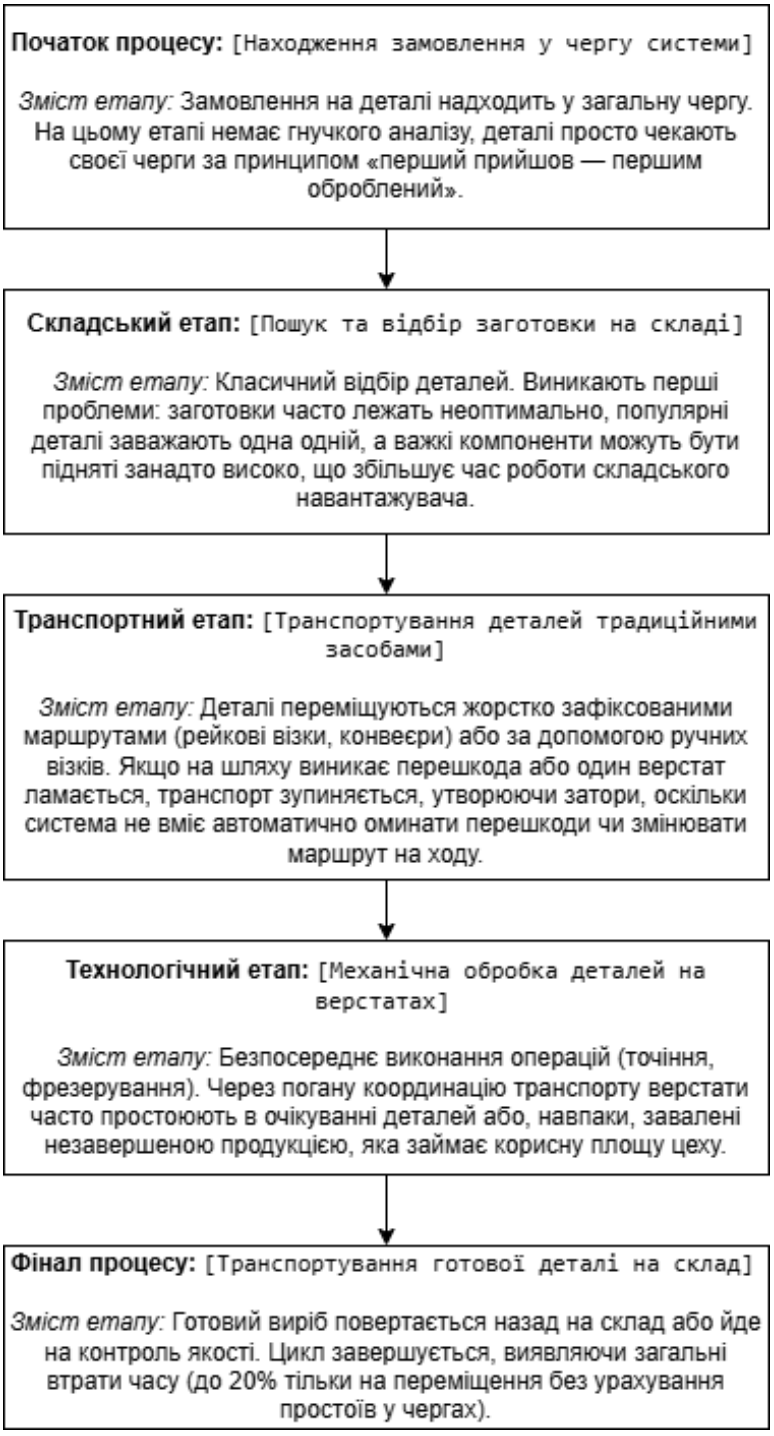


Рис. 1.1. Схема принципу роботи традиційної транспортно-складської системи на виробництв

1.2. Проблеми та недоліки традиційних методів керування потоками деталей

Для того щоб зрозуміти, чому сучасні підприємства активно переходять на автоматизацію та інтелектуальні системи керування, необхідно детально розібрати

проблеми, з якими стикається завод при використанні традиційних підходів. Коли логістичні потоки деталей координуються вручну диспетчерами або за допомогою жорстких, давно прописаних графіків, уся гнучкість виробництва зводиться нанівець. Будь-яке мінімальне відхилення від плану — поломка верстата, затримка постачання чи виявлення браку — викликає ефект доміно, через який починає гальмувати весь цех.

Перша і найпомітніша проблема традиційного керування полягає у великих втратах часу та накопиченні величезної кількості незавершеного виробництва на робочих місцях. Деталі просто стоять у чергах. У закордонній практиці цей час очікування та затримок у ланцюжку створення вартості описують терміном "**lead time**" (загальний час виконання процесу). У дослідженнях, що стосуються аналізу тривалості виробництва, цей чинник виділяють як один із найбільш критичних:

Виробники часто визначають час виконання (**lead time**) як загальний час, що минув від моменту отримання замовлення клієнта до моменту доставки продукції. Це охоплює всі види діяльності, що додають вартість (обробка, контроль), і час очікування (черги, транзит) у виробництві та ланцюжку поставок. Швидший час виконання означає більшу гнучкість, меншу кількість запасів і вищу задоволеність клієнтів [4].

При традиційних методах керування цей показник **lead time** зазвичай є дуже незадовільним саме через те, що логістичні операції очікування та транзиту займають до 80–90% від усього часу перебування деталі в цеху. Спроби розв'язати цю проблему «на око» або за допомогою паперових журналів диспетчера не дають результату, оскільки людина фізично не здатна прораховувати оптимальні маршрути для тисяч різнорідних деталей одночасно.

Другий серйозний недолік пов'язаний із жорсткістю та обмеженістю самих транспортних засобів, які використовуються на традиційних лініях. Найчастіше це стаціонарні конвеєри, рольганги або рейкові візки. Вони рухаються виключно по одній траєкторії. Якщо на такій лінії стається збій, робота зупиняється. Фахівці у сфері проєктування внутрішньоцехового транспорту зазначають:

Основними недоліками існуючих транспортних систем є обмежена маневреність та відсутність можливості швидкої перебудови маршруту в реальному часі. Використання жорстко зафіксованих траєкторій руху призводить до появи "пляшкових шийок" на виробництві, коли через зупинку чи поломку одного елемента системи блокується рух усього потоку деталей [2].

Тобто традиційний транспорт є абсолютно «сліпим» до поточної ситуації в цеху. Якщо перед рейковим візком поставити піддон або якщо на його шляху зупиниться інший навантажувач, він не зможе об'їхати перешкоду. Потрібно чекати, поки втрутиться людина, а тим часом верстати на наступній ділянці починають простоювати без заготовок.

Третя проблема криється всередині традиційної організації складського господарства. Склади на традиційних підприємствах часто працюють відокремлено від поточної ситуації на конвеєрі. Розміщення деталей на стелажах відбувається за спрощеними схемами, які абсолютно не враховують динаміку виробництва. У літературі з автоматизації складів чітко вказано, до чого це призводить на практиці:

Класичні підходи до складування часто призводять до нерівномірного розподілу найзатребуваніших позицій між зонами, надмірного навантаження окремих ділянок і необґрунтованого підйому важких компонентів на високі яруси. Це збільшує середній час відбору деталей та створює внутрішні затори у складській зоні [3].

Коли навантажувач або кран-штабелер витрачає забагато часу на те, щоб дістати важку заготовку з верхнього ярусу чи далекого кутка складу, транспортна система цеху починає працювати із запізненням. Виникає дисбаланс: склад не встигає видавати деталі із тією швидкістю, з якою їх готові обробляти сучасні верстати.

Четвертий недолік — це інформаційний розрив та відсутність синхронізації між матеріальним і документальним (інформаційним) потоками. У традиційних системах інформація про те, що деталь оброблена і готова до переміщення, передається із запізненням — через паперові накладні, змінні рапорти або ручне

введення даних в кінці робочої зміни. У посібниках з управління ланцюгами постачань цей ефект описується як головна причина хаосу:

Параметри результуючого потоку формуються в умовах незалежних управлінських впливів, що здійснюються послідовно в ланках логістичної мережі. Тому з погляду загальних цілей управління параметри результуючого потоку є спонтанними... Відсутність єдиного інформаційного простору викликає затримки в прийнятті рішень та призводить до надлишкового накопичення матеріальних запасів [1].

Через цю «спонтанність» параметрів диспетчер транспортного цеху дізнається про потребу в перевезенні деталей лише тоді, коли біля верстата вже фізично немає місця для складання готової продукції. Як наслідок, транспорт працює в режимі «гасіння пожеж», хаотично переїжджаючи від одного робочого місця до іншого, що суттєво збільшує холостий пробіг техніки та витрати на енергоносії.

Підсумовуючи аналіз традиційних методів, можна виділити їхні ключові недоліки:

1. Високий рівень незавершеного виробництва. Через погану координацію між ділянками цехи захаращуються деталями, які чекають своєї черги днями або тижнями [1, 4].

2. Низька швидкість реакції на збої. Традиційна система не здатна миттєво перерахувати логістичний маршрут, якщо зламався один із верстатів або запізнився транспортний засіб [2].

3. Неефективне використання площ і обладнання. Через затори на складах і жорсткі транспортні траєкторії дорогі верстати з програмним керуванням частину зміни простоюють, знижуючи загальну рентабельність виробництва [3].

4. Надмірні витрати на внутрішню логістику. Великий відсоток холостих ходів транспорту та зайві операції з переміщення і перескладання деталей збільшують собівартість готової продукції [2].

Усі ці недоліки доводять, що подальший розвиток виробничих систем неможливий без докорінної зміни принципів керування потоками деталей, що й зумовлює необхідність розробки автоматизованої та інтелектуальної системи.

Щоб краще зрозуміти, звідки беруться всі ці проблеми на заводі і як вони пов'язані між собою, пропоную поглянути на просту схему. Вона наочно показує, як технічні недоліки або затримки з паперами в результаті призводять до простою верстатів і зайвих витрат грошей. Весь цей ланцюжок причин і наслідків зображено на рисунку 1.2.

Рис. 1.2. Схема причинно-наслідкового аналізу недоліків традиційного керування

1.3. Порівняльний аналіз технічних засобів автоматизованого транспортування та складування

Коли підприємство вирішує відмовитися від застарілих ручних методів логістики, воно постає перед складним вибором: які саме засоби обрати для переміщення та зберігання деталей. Сьогодні на ринку автоматизації є безліч рішень — від класичних конвеєрів до розумних роботів-візків та роботизованих складських комплексів. Щоб зробити правильний вибір для проектованої системи, необхідно розібрати плюси та мінуси основних варіантів.

Усі технічні засоби внутрішньоцехової логістики діляться на дві великі групи: **транспортні** (які перевозять деталі між верстатами) та **складські** (які відповідають за зберігання та видачу заготовок).

Якщо говорити про транспорт, то класичним рішенням є **стаціонарні конвеєрні системи** (стрічкові, роликові чи ланцюгові). Вони чудово підходять для масового виробництва, де тисячі однакових деталей рухаються за одним маршрутом. Проте для сучасного приладобудування чи середньосерійного виробництва конвеєри стають проблемою через свою жорсткість. Вони займають забагато корисного простору цеху, їх складно переналаштувати під нову деталь, а в разі поломки однієї ланки зупиняється вся лінія.

Саме тому сучасні заводи переходять на мобільні та маневрені рішення — **автоматизовані транспортні візки**. Особливо перспективним напрямком є використання візків на спеціальних колесах, що дозволяють рухатися у будь-якому напрямку без зайвих розворотів. Науковці у своїх статтях детально описують переваги таких конструкцій:

Проведені дослідження та аналіз існуючих аналогів дозволили обґрунтувати доцільність створення автоматизованого транспортного візка на базі рушіїв Меканум, які забезпечують високу маневреність та точність позиціонування платформи за рахунок можливості всеспрямованого руху без зміни орієнтації корпусу в просторі [2].

Такі автоматизовані візки (AGV або AMR) не прив'язані до підлоги. Вони отримують команди через бездротову мережу і можуть самостійно об'їжджати перешкоди, змінюючи свій маршрут залежно від завантаженості верстатів.

Що стосується складської частини системи, то тут також відбувається технологічна еволюція. Традиційні склади, де водії навантажувачів шукають потрібний піддон за паперовими журналами, замінюються на автоматизовані системи пошуку та видачі. Головне завдання сучасного складу — не просто тримати в собі заготовки, а максимально швидко і без помилок видавати їх на транспортну лінію. Фахівці з автоматизації складського господарства зазначають:

Впровадження автоматизованих складських систем (AS/RS) у поєднанні зі спеціалізованими алгоритмами адресної оптимізації дозволяє суттєво скоротити час циклу пошуку та видачі вантажів, мінімізувати динамічні навантаження на конструкцію стелажів та повністю виключити помилки, викликані людським фактором [3].

Замість звичайних статичних стелажів на передових підприємствах встановлюють автоматизовані висотні системи зі спеціальними кранами-штабелерами або роботизованими шатлами. Вони самостійно піднімають та опускають потрібну деталь, орієнтуючись за штрих-кодами або RFID-мітками. Людина в цьому процесі взагалі не бере участі, що повністю виключає так званий «людський фактор», коли деталь випадково поставили не на ту полицю і потім шукають її усім цехом.

Таким чином, обираючи технічні засоби для проектованої системи, потрібно шукати баланс між їхньою вартістю, продуктивністю та гнучкістю. Поки що жоден засіб не є абсолютно ідеальним для всіх задач одночасно, тому їх часто комбінують у межах одного підприємства.

Щоб наочно порівняти між собою різні види автоматизованого транспорту та складів, а також побачити їхні сильні та слабкі сторони, глянемо на таблицю 1.1. Це допоможе обґрунтувати, чому для нашої системи автоматизації вибрано саме конкретні технічні рішення.

Таблиця 1.1. Порівняльний аналіз технічних засобів логістики

Технічний засіб	Головні переваги (Плюси)	Основні недоліки (Мінуси)	Найкраща сфера застосування	Посилання на джерело
Стаціонарні конвеєри (рольганги, стрічки)	Безперервна подача, дуже висока продуктивність на стабільних потоках.	Немає маневреності, займають багато місця, важко перепланувати цех.	Масове та великосерійне виробництво однотипних деталей.	[1, 2]
Автоматизовані візки (AGV/AMR)	Висока маневреність (особливо на колесах меканум), легка зміна маршрутів.	Обмежена вантажопідйомність, залежність від заряду акумулятора.	Гнучке середньосерійне виробництво, приладобудування.	[2]
Класичні склади з ручним пошуком	Низька вартість обладнання, простота організації процесу.	Повільний пошук деталей, часті помилки через людський фактор.	Невеликі майстерні або дрібносерійне виробництво.	[1, 3]
Автоматизовані висотні склади (AS/RS)	Максимальне використання висоти цеху, швидкий автоматичний пошук без помилок.	Дуже висока вартість обладнання та складне обслуговування.	Великі логістичні центри, сучасні автоматизовані заводи.	[3]

1.4. Огляд математичних методів та алгоритмічних рішень для оптимізації потоків

Для того щоб автоматизована транспортно-складська система працювала ефективно, недостатньо просто купити маневрені візки Меканум та сучасні стелажі. Потрібен математичний апарат, який буде керувати цим залізом у

реальному часі. Система має щомиті вирішувати складні завдання: яку деталь везти першою, як оминати затори в цеху, на яку саме полицю складу покласти вантаж, щоб потім не витрачати купу часу на його пошук. У науковій літературі для розв'язання таких задач використовуються різні класи математичних методів.

Перший великий клас — це класичні методи дослідження операцій та комбінаторної оптимізації. Сюди відносяться методи лінійного та динамічного програмування, а також теорія графів. Коли ми розглядаємо цех як мережу доріг, де верстати та склади є вузлами, задача пошуку найкоротшого шляху для транспортного візка стає класичною грою на графі. Проте, коли замовлень стає багато, класична математика починає давати збій через занадто довгі розрахунки. У дослідженнях складської логістики цей момент описується так:

Задачі оптимізації розміщення об'єктів у просторі автоматизованого складу належать до класу NP-важких задач комбінаторної оптимізації. Точні математичні методи (наприклад, метод гілок і меж або повний перебір) дозволяють знайти глобальний оптимум лише для невеликої розмірності задачі, тоді як при реальних обсягах виробництва їх використання обмежене через експоненціальне зростання обчислювальної складності [3].

Тобто, якщо на складі лежить всього 10 деталей, комп'ютер знайде ідеальну полицю за мікросекунду. Але якщо деталей тисячі, точний розрахунок триватиме годинами, що неприпустимо для конвеєра, який працює в реальному часі.

Саме тому **другим класом** методів є евристичні та метаевристичні алгоритми (генетичні алгоритми, алгоритми мурашиної колонії, імітації відпалу тощо). Вони не шукають «ідеальне» рішення цілу вічність, а знаходять «достатньо хороше і практичне» рішення за кілька секунд. Наприклад, мурашині алгоритми чудово копіюють поведінку живих мурах, які знаходять найкоротшу стежку до їжі за допомогою феромонів. У транспортній логістиці це допомагає візкам динамічно оминати аварійні ділянки чи затори.

Третій, дуже важливий для нашої теми клас — це **теорія масового обслуговування (ТМО)** та використання пріоритетних черг. Оскільки деталь у процесі обробки постійно чекає (то на складі, то перед верстатом, то в очікуванні

візка), її рух можна описати як проходження через систему черг. Якщо керувати цими чергами хаотично (за принципом «хто перший прийшов»), то термінові деталі будуть стояти поруч із другорядними, гальмуючи збирання готового приладу.

У посібниках з логістики виробничих систем підкреслюється, що керування чергами є основою для стабільної роботи:

Для забезпечення безперервності виробничого процесу необхідно реалізувати чіткі правила формування черг та обслуговування матеріальних потоків на технологічних ділянках. Застосування динамічних пріоритетів дозволяє гнучко перерозподіляти ресурси транспортної системи користю тих замовлень, які є найбільш критичними з погляду загального планового графіка підприємства [1].

Впровадження пріоритетних черг дозволяє кожній деталі в системі присвоїти свій «ваговий коефіцієнт» (важливість). Якщо верстат звільняється, система автоматично витягує з черги не просто випадкову деталь, а саме ту, затримка якої може зупинити роботу наступних цехів.

Четвертий напрямок — це імітаційне моделювання. Оскільки виробничий цех є занадто складною і стохастичною (випадковою) системою, прорахувати все формулами на папері неможливо. Завжди є чинник випадковості: верстат може зламатися, інструмент — затупитися, а транспортний візок — затриматися на повороті. У статтях, що присвячені моделюванню логістики, зазначається:

Імітаційне моделювання є найбільш ефективним інструментом для аналізу динаміки транспортних потоків в умовах випадкових збурень. Воно дозволяє ще на етапі проектування виявити потенційні місця виникнення заторів ("пляшкові шийки"), оцінити завантаженість транспортних засобів та перевірити працездатність алгоритмів керування без ризику зупинки реального виробництва [2].

Завдяки імітаційному моделюванню можна створити «цифрового двійника» нашого цеху, запустити туди віртуальні деталі та перевірити, як алгоритми з пріоритетними чергами справлятимуться із навантаженням.

Таким чином, для нашої дипломної роботи найкращим рішенням є **комбінований підхід**. Ми візьмемо елементи **теорії масового обслуговування** (для математичного опису черг деталей) та поєднаємо їх із **динамічними пріоритетами**, а перевірку ефективності проведемо за допомогою програмного моделювання.

Для того щоб чітко структурувати всі існуючі математичні підходи та побачити, яке місце серед них посідають алгоритми пріоритетних черг і моделювання, пропоную розглянути загальну класифікацію методів оптимізації потоків, що наведена на рисунку 1.3.

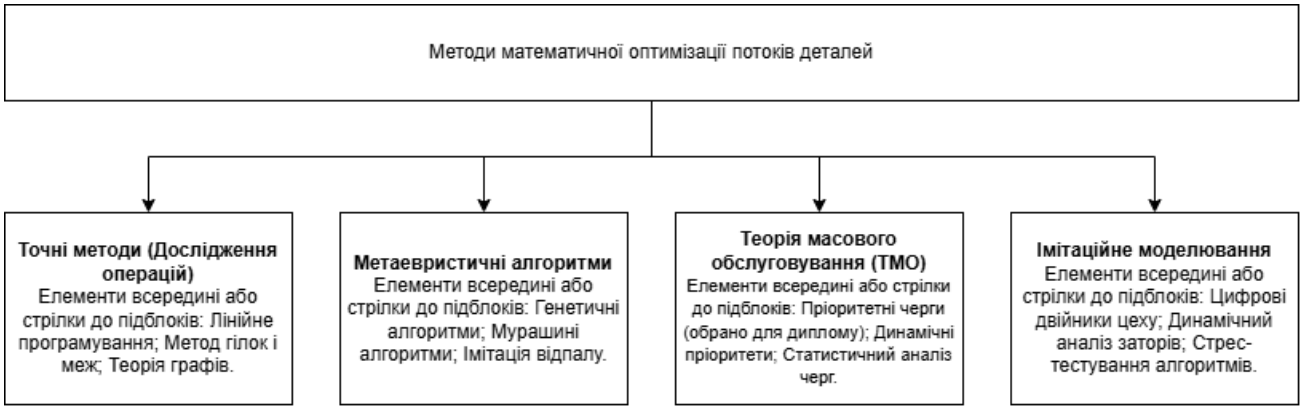


Рис. 1.3. Класифікація математичних методів оптимізації логістичних потоків

1.5. Постановка задачі дипломної роботи, визначення цільової функції та обмежень

Підводячи підсумки аналізу, проведеного у першому розділі, можна впевнено сказати: сучасне автоматизоване виробництво вже не може ефективно функціонувати за старими, жорстко зафіксованими алгоритмами конвеєрів. Ми з’ясували, що класичні підходи (FIFO – «першим прийшов — першим обслужений», фіксовані маршрути) створюють затори, змушують дорогі складальні верстати простоювати в очікуванні заготовок або, навпаки, перевантажують проміжні склади. Технічні засоби — від звичайних рольгангів до розумних роботів та транспортних візків (AGV) — мають колосальний потенціал,

але без гнучкого «мозку», тобто інтелектуальної системи керування, вони залишаються просто набором заліза.

Щоб чітко зрозуміти, куди рухатися далі, коротко розберемо переваги та недоліки існуючих сьогодні автоматизованих транспортних ліній:

Традиційні (жорсткі) транспортні лінії:

Плюси: висока надійність, просте проектування та низька вартість базової автоматизації. Якщо завод роками випускає одну й ту саму деталь без жодних змін, така система працює стабільно.

Мінуси: абсолютна негнучкість. Будь-яка зміна специфікації моделі, поломка на одній із дільниць або затримка постачання сировини паралізує всю лінію. Система не вміє самостійно перерозподіляти потоки в реальному часі.

Сучасні гнучкі виробничі системи (ГВС) з базовою автоматизацією:

Плюси: використання датчиків (наприклад, зчитувачів штрих-кодів чи RFID-міток) дозволяє ідентифікувати деталі та розподіляти їх за напрямками.

Мінуси: зазвичай керування відбувається на основі статичних правил. Система бачить, яка деталь перед нею, але не враховує поточний стан усього цеху (наприклад, де саме зараз критичний дефіцит компонентів, а де накопичився брак).

Запропоноване рішення: Лінія з багаторівневою системою пріоритетів.

У межах цієї дипломної роботи пропонується концептуально нове рішення: автоматизована транспортна лінія з багаторівневою динамічною системою пріоритетів.

Уявімо реальний виробничий майданчик. У нас є загальний цех, де виготовляються деталі, та три окремі дільниці, на яких збираються схожі за структурою моделі виробів. Транспортна система будується за наступною логікою руху та аналізу:

Етап виробництва та первинного контролю: Щойно деталь виготовлено, вона потрапляє на етап перевірки відповідності її розмірів. Тут автоматика перевіряє геометричні параметри та потрібні властивості. Якщо виявлено

відхилення від креслення, система діє гнучко: аналізує допуски і приймає рішення — або перенаправити деталь на іншу дільницю (де такий допуск за специфікацією вважається допустимим), або повернути її назад на доопрацювання для виправлення розміру, або ж повністю відправити на переробку.

Етап аналізу та специфікації: Якщо деталь пройшла контроль розмірів, автоматизована система оцінює її специфікацію — чи є деталь уніфікованою (придатною для кількох моделей), чи індивідуальною (створеною під конкретну модель на одній з дільниць).

Багаторівневий алгоритм розподілу: Коли постає питання, куди саме спрямувати деталь, система починає прораховувати пріоритети складальних дільниць на основі живих даних:

Перший рівень (Пріоритет за потребою): Система перевіряє поточну кількість деталей на кожній дільниці. Якщо деталь закінчилася чи добігає кінця, статус цієї дільниці автоматично стає максимально пріоритетним, щоб уникнути зупинки збірки.

Другий рівень (Стратегічний пріоритет дільниці): Якщо деталь потрібна одночасно кільком дільницям, вмикається пріоритетність самих дільниць (черговість: куди треба постачати в першу чергу, в другу, в третю тощо відповідно до загального плану).

У результаті, якщо потреба виникає одночасно у двох місцях, автоматизована лінія спочатку транспортує деталь на дільницю з вищим стратегічним пріоритетом, а наступну — на ту, що нижча за рейтингом.

Зворотний зв'язок: Наприкінці шляху вся інформація про кількість переданих деталей миттєво відправляється до загального аналізатора. Це дозволяє системі постійно зберігати в пам'яті актуальні значення потреб по кожній збірці.

Переваги мого варіанта порівняно з існуючими:

Висока адаптивність: Система не працює наосліп, вона динамічно реагує на реальний дефіцит деталей на кожному робочому місці.

Зниження відсотків відходів: Завдяки «розумному» контролю розмірів деталь не бракується одразу, а отримує другий шанс, якщо підходить під специфікацію іншої складальної дільниці.

Повний інформаційний контроль: Завдяки аналізатору, який постійно оновлює дані, система завжди знає та показує точний стан виробництва.

Можливі недоліки та ризики:

Складність алгоритмічної логіки: Побудова багаторівневих черг вимагає надійної програмної логіки, яку складніше налаштовувати при первинному запуску.

Залежність від надійності датчиків: Оскільки система повністю покладається на автоматичний контроль розмірів та підрахунок деталей у реальному часі, будь-яка відмова вимірювального пристрою чи датчика на лінії може призвести до хибного розподілу потоків.

Концептуальна постановка задачі:

Таким чином, задача дипломної роботи полягає у проєктуванні та дослідженні автоматизованої системи керування, яка повинна забезпечити оптимальний розподіл матеріальних потоків деталей між складальними дільницями.

Визначення цільової функції:

Основним критерієм ефективності (цільовою функцією) проєктованої системи є мінімізація часу простоїв складальних дільниць та оптимізація загального часу транспортування деталей. Система керування повинна розподіляти потоки так, щоб сумарні затримки виробництва через відсутність заготовок прагнули до нуля.

Визначення системи обмежень:

Для успішного вирішення завдання система повинна функціонувати в межах наступних технологічних обмежень:

Обмеження на розподіл: Кожна виготовлена деталь може бути направлена в один момент часу лише на один маршрут (до однієї дільниці, на доопрацювання або в брак).

Обмеження за специфікацією: Направлення деталі на складальну дільницю можливе лише за умови її повної відповідності геометричним допускам та технічним вимогам цієї дільниці.

Обмеження на місткість буферів: Кількість деталей, що накопичуються в проміжних зонах або накопичувачах перед дільницями, не повинна перевищувати їхню максимальну місткість, щоб не спричинити фізичного блокування транспортних шляхів лінії.

Розробка структури такої системи, її алгоритмічного та програмного забезпечення для реалізації описаної багаторівневої логіки і стане основним завданням наступних розділів дипломної роботи.

Висновки до розділу 1

У першому розділі дипломної роботи проведено комплексний аналіз сучасного стану, проблем та перспектив розвитку автоматизованих систем керування транспортними потоками деталей у промисловому виробництві. За результатами проведеного аналізу та літературного огляду можна зробити такі висновки:

1. Сучасний етап розвитку виробництва (Індустрія 4.0) вимагає переходу від традиційних жорстких конвеєрних ліній до гнучких автоматизованих систем. Ефективність сучасного цеху безпосередньо залежить не лише від швидкості роботи окремого технологічного обладнання, а й від інтелектуального рівня системи керування логістичними та транспортними потоками всередині підприємства.

2. Традиційні підходи до розподілу матеріальних потоків (такі як статичне маршрутизування або черги за принципом FIFO) мають критичні недоліки. Вони виявляються нездатними адаптуватися до динамічних змін на виробництві: коливань темпу роботи складальних дільниць, появи бракованої продукції чи раптової зміни пріоритетів виробничого плану. Це призводить до виникнення заторів на транспортних лініях та тривалих простоїв обладнання.

3. Огляд існуючих технічних рішень та математичних методів моделювання черг показав, що найбільш перспективним напрямком автоматизації є створення адаптивних систем керування. Проте більшість розглянутих систем спираються на статичні правила вибору маршрутів і не здатні комплексно оцінювати поточний стан та потреби кількох складальних зон одночасно.

4. Обґрунтовано концепцію створення нової автоматизованої транспортної лінії, що базується на впровадженні багаторівневої системи динамічних пріоритетів. Особливістю запропонованого підходу є інтеграція етапу інтелектуального геометричного контролю (з можливістю гнучкого перенаправлення деталей на альтернативні дільниці) та безперервного моніторингу реального дефіциту компонентів на робочих місцях із передачею даних до центрального аналізатора.

5. На основі запропонованого концепту сформульовано загальну постановку задачі дипломної роботи. Визначено цільову функцію, спрямовану на мінімізацію часу простоїв складальних дільниць та оптимізацію часу транспортування, а також встановлено систему технологічних обмежень (за специфікацією деталей, місткістю проміжних буферів та умов розподілу потоків).

Отримані в межах аналітичного огляду висновки та сформульовані критерії ефективності є базою для розробки структурно-функціональної схеми, математичного опису алгоритмів пріоритетних черг та програмного моделювання системи у наступних розділах роботи.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ КЕРУВАННЯ ПОТОКАМИ ДЕТАЛЕЙ

2.1.Структурно-функціональна схема автоматизованої системи керування

Для того щоб спроєктувати ефективну автоматизовану систему керування транспортними потоками деталей (АСК ТПД), спочатку потрібно чітко розібратися, з яких частин вона складатиметься та як ці частини взаємодіятимуть між собою. Найкращий спосіб це зробити — побудувати структурно-функціональну схему. Вона дає змогу розділити загальний складний процес на окремі керовані блоки, показати шлях руху фізичних компонентів (матеріальний потік) та визначити, як саме між елементами лінії циркулює керуюча інформація (інформаційний потік) [1].

Оскільки наша лінія працює в умовах випадкового часу надходження деталей та різної тривалості операцій, її доцільно розглядати як систему масового обслуговування (СМО). У таких системах правильна побудова структури зв'язків допомагає оптимізувати параметри потоку заявок та обслуговування з метою мінімізації черг і затримок [5]. Розроблена структурно-функціональна схема нашої АСК ТПД представлена на рисунку 2.1.

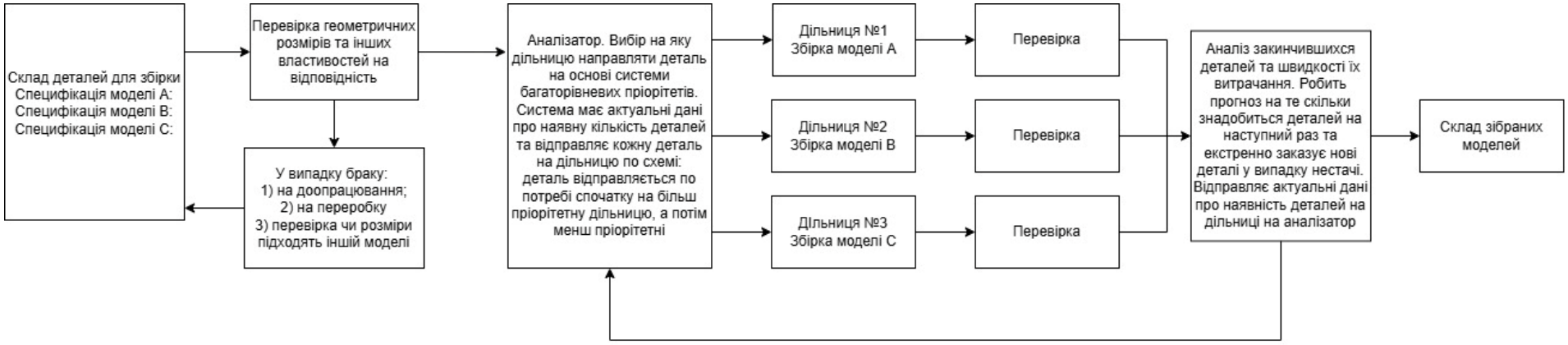


Рис. 2.1 Структурно-функціональна схема АСК ТПД

Розглянемо детально логіку роботи системи та призначення кожного функціонального блока, що зображений на схемі, за напрямком руху матеріального потоку:

Склад деталей: Це початкова точка нашої системи. На відміну від простого виробничого цеху, тут зберігається чітко визначений список деталей під конкретні моделі виробів, а також фіксується точна кількість деталей, яку завіз постачальник. Цей блок виступає в ролі джерела заявок для нашої системи масового обслуговування [5].

Пост контролю розмірів: Зі складу деталі потрапляють на контроль відповідності. Це перший критичний вузол автоматизації. Датчики зчитують реальні розміри деталі, і якщо виявляється відхилення від норми, система оперативно вирішує: чи можна перенаправити деталь на іншу складальну дільницю (де вимоги до точності нижчі), чи повернути назад для механічного виправлення, чи списати в остаточний брак. Оскільки цей блок безпосередньо впливає на маршрутизацію транспорту у виробництві, його логіка спирається на принципи оптимізації транспортних потоків деталей [2].

Блок аналізу за специфікацією: Тут система порівнює параметри придатної деталі з вимогами технологічних карт і визначає, на яку саме з трьох складальних дільниць її можна відправити за технічними характеристиками.

Блок багаторівневих пріоритетів: Це інтелектуальне «серце» транспортної системи. Його головне завдання — повністю усунути простій складальних дільниць без роботи, адже кожна хвилина зупинки — це прямі збитки для підприємства [1]. Блок аналізує потреби кожної дільниці в реальному часі. Окрім того, сюди закладено важливий математичний принцип: моделі виробів, які за технологією збираються найдовше, закріплюються за дільницями з найвищим пріоритетом. Такий підхід дозволяє збалансувати загальний час роботи між усіма трьома дільницями та уникнути ситуації, коли один пост перевантажений, а інші відпочивають [3].

Складальні дільниці (№1, №2, №3): Безпосередні робочі місця, де зі схожих деталей збираються різні моделі продукції. Перед кожною дільницею є свій буфер (накопичувальна черга) обмеженої місткості, що моделюється відповідно до теорії черг у гнучких виробничих системах [6].

Перевірка збірок: Новий етап контролю якості, куди готові вироби потрапляють одразу після складальних дільниць. Тут перевіряється правильність збирання моделі перед тим, як відправити її споживачу.

Аналіз актуальної кількості деталей та прогнозування: Сюди безперервно надходит інформація про кількість успішно зібраних і перевірених виробів. На основі цих даних блок автоматично вираховує, скільки деталей було витрачено і скільки реально залишилося на складі. У реальному житті, навіть якщо постачальник завіз ідеальну кількість деталей під замовлення, частина з них може піти в брак на етапі геометричного контролю. Тому, якщо система бачить дефіцит через відсіювання браку, вона автоматично формує замовлення на доставку недостачі. Окрім цього, блок виконує функцію прогнозування — він прораховує темпи збирання і заздалегідь каже, скільки і яких деталей знадобиться кожній дільниці на наступні робочі зміни [1, 3].

Склад готової продукції: Фінальна точка, куди переміщуються повністю готові та перевірені вироби для подальшого пакування та відвантаження.

Завдяки такій структурі із розвиненими зворотними зв'язками (від аналізатора та прогнозувальника назад до блоку пріоритетів і складу деталей), інформація в системі оновлюється кожну секунду. Наша АСК ТПД діє не як жорсткий заздалегідь запрограмований конвеєр, а як гнучкий організм, що самостійно підлаштовується під поточний стан виробництва, страхує від браку, оптимізує логістичні маршрути [2] і надійно захищає дільниці від простоїв у чергах буферів [6].

2.2. Математична модель багаторівневої системи керування з використанням пріоритетних черг

Для формалізації та підвищення ефективності функціонування автоматизованої системи керування транспортними потоками деталей (АСК ТПД) необхідне строге математичне обґрунтування. Оскільки надходження заготовок на лінію та тривалість їх технологічної обробки (перевірки розмірів, аналізу специфікацій, безпосереднього збирання) мають випадковий характер, виробниче середовище доцільно розглядати як класичну систему масового обслуговування (СМО) [1].

Згідно з методологічними засадами дослідження стаціонарних багатоканальних систем масового обслуговування, математичний опис функціонування лінії базується на розрахунку інтенсивностей потоків та ймовірностей станів системи. Як зазначено в теоретичних основах проектування логістичних процесів у виробничих системах, інтенсивність надходження деталей на вхід системи λ (серівна кількість заявок за одиницю часу) визначається як величина, обернена до середнього часу між надходженнями двох послідовних деталей [1]:

$$\lambda = \frac{1}{t_{\text{надход}}} \tag{2.1}$$

Аналогічно, спроможність системи щодо обробки деталей на постах контролю або складальних дільницях характеризується інтенсивністю обслуговування μ. Відповідно до технічної документації та розрахункових моделей виробничих потоків, цей параметр розраховується через середній час виконання однієї технологічної операції [2]:

$$\mu = \frac{1}{t_{\text{обс}}} \tag{2.2}$$

Для забезпечення стабільної роботи автоматизованої лінії без безмежного зростання черг і перевантаження буферів накопичувачів необхідно виконувати умову стаціонарності виробничого процесу. Загальний коефіцієнт завантаження системи (або інтенсивність навантаження) для n-канальної СМО відображає відношення інтенсивності надходження до сумарної пропускної спроможності каналів [5]:

$$\rho = \frac{\lambda}{n\mu} < 1 \tag{2.3}$$

У проєктованій системі функціонують три паралельні складальні дільниці ($M = 3$), які утворюють багатоканальну структуру. Розподіл ймовірностей станів такої системи у стаціонарному режимі, коли немає залежання від початкових умов, описується класичними рівняннями Ерланга. При цьому ймовірність того, що всі канали обслуговування та проміжні накопичувачі є вільними від деталей, визначається за наступною залежністю [4]:

$$P_0 = \left[\sum_{h=0}^{n} \frac{(n\rho)^k}{k!} + \frac{(n\rho)^{n+1}}{n!(1-\rho)}\right] \quad (2.4)$$

Відповідно, ймовірність застаання системи у стані, коли завантажені всі n каналів, визначає ризик виникнення затримок та формування технологічної черги перед дільницями. Ця характеристика є критичною для оцінки загальної пропускної спроможності автоматизованої лінії [5]:

$$P_{\text{очік}} = \frac{(n\rho)^n}{n!(1-\rho)} P_0 \quad (2.5)$$

Традиційні підходи до керування потоками в СМО базуються на дисципліні обслуговування «перший прийшов — перший пішов» (FIFO). Проте в умовах багатономенклатурного середньосерійного виробництва приладів така схема призводить до тривалих простоїв обладнання через нерівномірність постачання уніфікованих деталей та специфічних компонентів для окремих моделей. Для мінімізації втрат ефективності впроваджується багаторівнева система пріоритетів, де кожній деталі, що надходить у систему, присвоюється динамічний інтегральний пріоритет $W(i,j)$ для спрямування на j-ту дільницю [3]:

$$W(i,j) = \alpha P_{\text{ПрПотр}}(j) + \beta P_{\text{ПрДет}}(i) + \gamma P_{\text{ПрДільн}}(j) \quad (2.6)$$

де:

$P_{\text{ПрПотр}}(j)$ — пріоритет за поточною потребою j-ї дільниці у конкретній деталі (зростає, якщо кількість деталей у буфері дільниці наближається до критичного мінімуму);

$P_{\text{ПрДет}}(i)$ — категорійний пріоритет самої деталі (уніфіковані деталі загального призначення мають один рівень, індивідуальні чи дефіцитні компоненти під конкретну модель — вищий);

$P_{\text{ПрДільн}}(j)$ — базовий технологічний пріоритет складальної дільниці у загальному виробничому циклі цеху;

$\alpha,\ \beta,\ \gamma$ — вагові коефіцієнти, що визначають ступінь впливу кожного чинника та налаштовуються залежно від поточного виробничого плану.

Введення пріоритетних черг суттєво трансформує математичну модель розрахунку середнього часу очікування деталі в черзі. Для системи з відносними пріоритетами (коли обслуговування поточної деталі не переривається, а пріоритет враховується лише в момент вибору наступної деталі з накопичувача), середній час очікування для деталі k-го пріоритетного класу визначається за формулою [5]:

$$t_{\text{очік}}^{k} = \frac{\sum_{s=1}^{R} \lambda t_{\text{обс},s}^{2}}{2\left(1-\sum_{s=1}^{k-1} \rho_s\right)-\left(1-\sum_{s=1}^{k} \rho_s\right)} \tag{2.7}$$

де:

R — загальна кількість виділених класів пріоритетів у системі;

λ_s — інтенсивність надходження деталей s-го класу пріоритету;

$t_{\text{обс},s}$ — середній час обслуговування (обробки чи контролю) деталей s-го класу.

Аналіз наведеної залежності показує, що правильний розподіл деталей за категоріями дозволяє значно знизити час очікування для найбільш критичних компонентів, запобігаючи зупинкам складальних конвеєрів.

У випадку виявлення невідповідності геометричних параметрів деталі на етапі автоматизованого контролю розмірів, матеріальний потік розгалужується. Ймовірність направлення деталі на повторну доробку, утилізацію (брак) або на іншу складальну дільницю, де відхилення розмірів є допустимим за специфікацією, описується через матрицю перехідних ймовірностей СМО. Сумарний потік заявок, що циркулює між постами контролю та накопичувачами, підпорядковується балансовим рівнянням для відкритих мереж обслуговування [3]:

$$\Lambda_j = \lambda_j + \sum_{k=1}^{K} \Lambda_k p_{kj} \tag{2.8}$$

де:

Λ_j — повна (сумарна) інтенсивність потоку деталей через j-й елемент АСК ТПД з урахуванням повернень та рециркуляції;

λ_j — інтенсивність первинного надходження деталей із заготовчого цеху;

p_{kj} — ймовірність переходу деталі від k-го технологічного вузла до j-го (визначається на основі аналізу специфікацій та результатів вимірювань датчиків).

Для реалізації оптимізаційного алгоритму розподілу деталей аналізатор системи постійно накопичує інформацію про поточний стан буферів. Загальна цільова функція управління транспортними потоками деталей формується як багатокритеріальна задача, спрямована на одночасне розв'язання двох протилежних задач: мінімізацію часу транспортування заготовок та повне виключення простоїв робочих місць на складальних дільницях [2]:

$$F = \min \left[\begin{matrix} \sum_{i=1}^{N} \quad \sum_{j=1}^{M} \quad t_{\text{трансп}}(i,j) \cdot x(i,j) + \\ \sum_{j=1}^{M} \Delta t_{\text{простой}}(j) \cdot W_{\text{ПоточПр}}(j) \end{matrix} \right] \tag{2.9}$$

де:

N — загальна кількість деталей, що проходять через лінію протягом поточної зміни;

M — кількість складальних дільниць ($M = 3$);

$t_{\text{трансп}}(i,j)$ — час безпосереднього транспортування i-ї деталі до j-ї дільниці за допомогою автоматизованих засобів;

$x(i,j)$ — булева змінна керування ($x(i,j) = 1$, якщо систему прийнято рішення направити i-ту деталь на j-ту дільницю, та $x(i,j) = 0$ в іншому випадку);

$\Delta t_{\text{простой}(j)}$ — час вимушеного простою j-ї дільниці через відсутність необхідних деталей;

$W_{\text{ПоточПр}(j)}$ — ваговий коефіцієнт (поточний пріоритет) дільниці, отриманий з блоку багаторівневих пріоритетів.

Наведена математична модель є основою для побудови керуючих алгоритмів та розробки програмного забезпечення АСК ТПД, що дозволяє гнучко адаптувати параметри черг до динамічних змін у реальному часі.

2.3. Алгоритм керування з врахуванням пріоритетних черг

На основі розробленої структурно-функціональної схеми та математичної моделі необхідно сформувати алгоритм керування транспортними потоками деталей. Основною особливістю запропонованого алгоритму є використання багаторівневої системи пріоритетних черг, яка дозволяє приймати рішення про маршрутизацію деталей не за класичним принципом FIFO («першим прийшов — першим обслужений»), а з урахуванням поточних потреб складальних дільниць, наявності запасів у буферних накопичувачах та стратегічної важливості кожної виробничої дільниці.

Традиційні підходи до розподілу матеріальних елементів у межах цеху мають суттєві обмеження гнучкості, оскільки класичні підходи до складування часто призводять до нерівномірного розподілу найзатребуваніших позицій між зонами, надмірного навантаження окремих ділянок і необґрунтованого підйому важких компонентів на високі яруси [3]. Більше того, коли логістичні потоки деталей координуються лінійними методами, це викликає значні затримки в прийнятті рішень та призводить до надлишкового накопичення матеріальних запасів [1]. Для уникнення цих явищ розроблений алгоритм орієнтований на гнучке динамічне реагування.

Робота алгоритму починається з моменту надходження нової деталі з цеху виготовлення до автоматизованої транспортної системи (етап «Початок» та блок «Склад деталей»). Після реєстрації деталі виконується контроль її геометричних параметрів на спеціалізованому посту. Тут відбувається «перевірка браку», що є критично важливим для забезпечення якості збірки. Якщо деталь відповідає встановленим допускам та вимогам збірки, система переходить до аналізу специфікації деталі та направляє її до розподільного пункту.

Але у разі виявлення відхилень від геометричних стандартів система гнучко реагує на цей стан: деталь відправляється на доопрацювання або переробку, або, якщо деталь підходить до іншої моделі, то змінює шлях до іншої дільниці, де такі геометричні відхилення є допустимими та не впливають на якість фінального

приладу. Це дозволяє уникнути передчасного вибраковування матеріалів і знижує виробничі втрати.

Для деталей, які успішно пройшли контроль, у розподільному пункті виконується визначення можливих напрямків транспортування. На цьому етапі блок-аналізатор здійснює «аналіз, на яку дільницю направити деталь; перевірка наявної кількості деталей на дільницях, визначення, де потрібна яка деталь». Система зчитує інформацію про поточний стан усіх трьох складальних дільниць. Це необхідно тому, що управління ланцюгами постачань (Supply Chain Management (SCM)) – це організація, планування, контроль та виконання товарного потоку, від проєктування і закупівель через виробництво та розподіл до кінцевого споживача відповідно до вимог ринку до ефективності витрат [1].

Далі алгоритм здійснює перевірку типу компонента через логічний розгалужувач: «Деталь уніфікована чи індивідуальна?». Обробка потоку заявок на цьому кроці суттєво відрізняється залежно від результату перевірки:

Якщо деталь визначена як **індивідуальна** (тобто призначена суворо для конкретної моделі, що збирається на одній визначеній дільниці), вона спрямовується «без черги на дільницю по потреби». Це мінімізує час перебування унікальних деталей у транзитних зонах і виключає простої специфічного обладнання.

Якщо деталь є **уніфікованою** (може бути одночасно використана на кількох дільницях, де збираються схожі моделі виробів), виникає ймовірність конфлікту запитів. Для вирішення таких ситуацій алгоритм використовує інтегральні вагові коефіцієнти, обчислені в математичній моделі. Уніфікована деталь «йде спочатку на ту дільницю, що вища пріоритетом та запросила цю деталь».

Пріоритетність дільниць формується багаторівневим контуром аналізу. По-перше, враховується пріоритет за поточною потребою: якщо на певній дільниці критично вичерпався буферний накопичувач і виникає загроза зупинки лінії, її запит отримує найвищий статус. По-друге, враховується стратегічна пріоритетність самої дільниці в загальному плані цеху (черговість виконання замовлень: перша черга, друга черга тощо). Правильна побудова структури зв'язків

допомагає оптимізувати параметри потоку заявок та обслуговування з метою мінімізації черг і затримок [2]. Динамічний перерахунок пріоритетів запобігає виникненню ситуацій, коли в деякі періоди часу на вході серверу накопичуються вимоги, які утворюють чергу на обслуговування, в інші ж періоди система працює з недовантаженням [5].

Після успішного розподілу та доставки компонента на відповідну дільницю відбувається безпосередня «збірка деталей у готовий продукт на дільниці», за якою слідує обов'язкова «перевірка якості збірки».

Завершальним етапом алгоритму є робота контуру зворотного зв'язку в автоматизованій системі. Проводиться «підрахунок використаних деталей, прогноз скільки залишилось на складі», а також детальний аналіз динаміки споживання. Вся отримана інформація миттєво передається назад до блока-аналізатора для актуалізації значень кількості деталей. Це дозволяє підтримувати базу даних потреб у реальному часі та забезпечує високу адаптивність системи до випадкових змін інтенсивності виробництва.

Таким чином, запропонований алгоритм забезпечує ефективне динамічне керування транспортними потоками в режимі реального часу та дозволяє мінімізувати простої складальних дільниць за рахунок пріоритетного забезпечення найбільш критичних виробничих операцій.

Схема алгоритму функціонування автоматизованої системи керування транспортними потоками деталей наведена на рисунку 2.2.

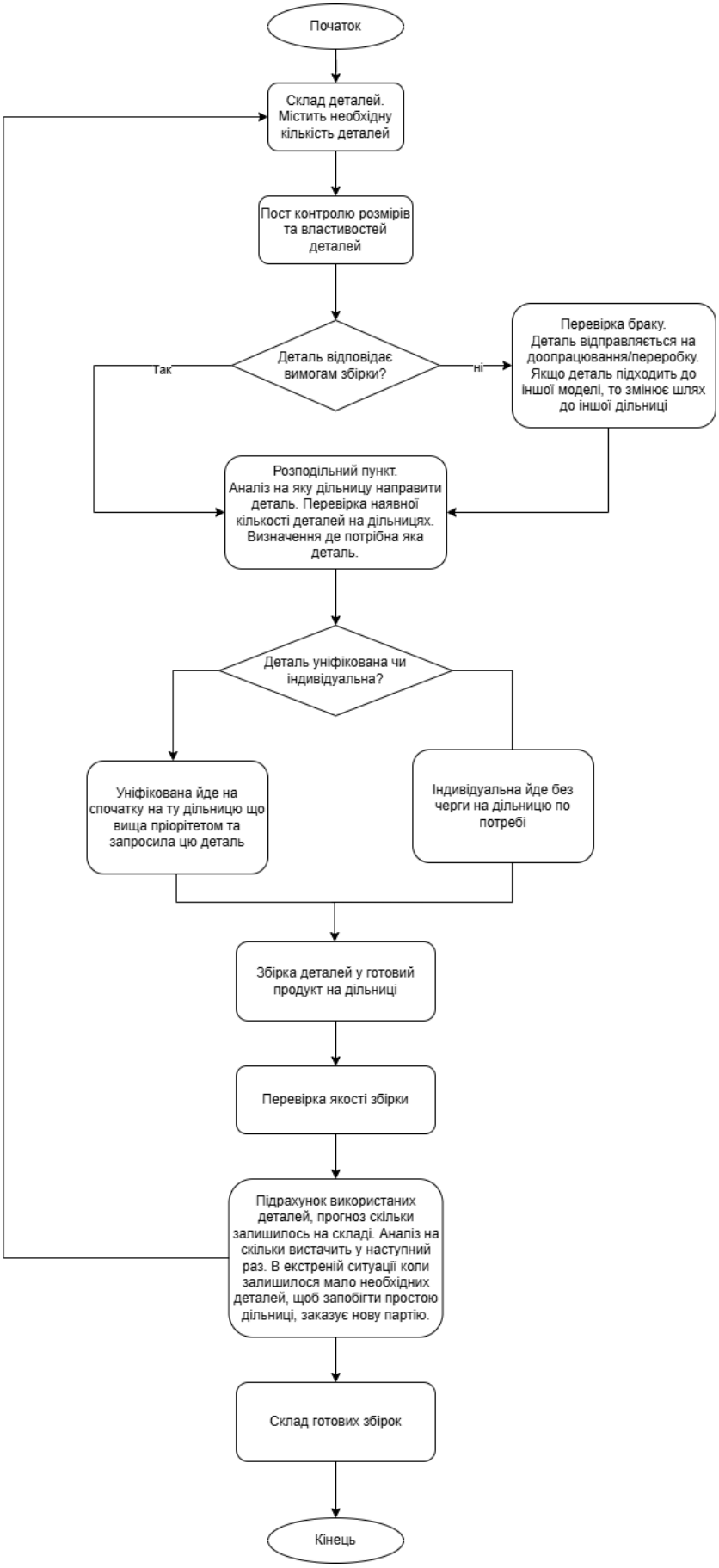


Рис. 2.2. Алгоритм керування з врахуванням пріоритетних черг

Висновки до розділу

У межах другого розділу було проведено комплексне проєктування автоматизованої системи керування транспортними потоками деталей (АСК ТПД), що дозволило сформувати системний підхід до організації внутрішньовиробничої логістики.

Ключовими результатами виконаної роботи є:

1. **Структурно-функціональна схема:** На основі побудованої схеми вдалося декомпозувати складний виробничий процес на окремі керовані блоки. Це дало змогу чітко розмежувати шляхи руху матеріальних потоків та контури передачі керуючої інформації, що є критично важливим для уникнення заторів на транспортних лініях.
2. **Математична модель:** Розроблена модель багаторівневої системи керування з використанням пріоритетних черг дозволила формалізувати динаміку виробничого середовища як системи масового обслуговування. Це дало базу для оцінки параметрів потоку заявок та обслуговування, спрямовану на мінімізацію затримок.
3. **Алгоритм керування:** Створений алгоритм керування реалізує принцип динамічного перерозподілу деталей між складальними дільницями. Завдяки впровадженню інтегральних пріоритетів та контуру зворотного зв’язку (аналізатора актуальних потреб), система здатна ефективно адаптуватися до змін інтенсивності виробництва у режимі реального часу, забезпечуючи пріоритетне постачання найбільш критичних компонентів.

Отже, розроблене науково-технічне забезпечення є надійним фундаментом для подальшої програмної реалізації системи. Спроєктована модель та алгоритми створюють необхідну базу для проведення комп’ютерного моделювання, що дозволить перевірити ефективність запропонованих рішень у реальних умовах автоматизованого виробництва перед їх безпосереднім впровадженням.

РОЗДІЛ 3.
РОЗРОБКА ВИДІВ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ КЕРУВАННЯ

Для успішної реалізації автоматизованої системи керування транспортними потоками деталей та імітаційного моделювання процесів розподілу компонентів між складальними дільницями критично важливим є обґрунтований вибір інструментарію розробки [1]. Специфіка досліджуваної предметної області вимагає обробки інформації в реальному часі, моделювання логістичних операцій, управління базою даних та наявності інтуїтивно зрозумілого інтерфейсу оператора [1, 2].

У якості базової мови програмування для розробки системи було обрано Python. Цей вибір обумовлений її високою ефективністю для швидкого прототипування імітаційних моделей, кросплатформеністю, лаконічністю синтаксису та наявністю потужної екосистеми спеціалізованих стандартних і зовнішніх бібліотек [7, 8].

В наступних розділах наведено детальний аналіз та обґрунтування використання конкретних програмних засобів, інструментів локального збереження даних та спеціалізованих модулів, які складають технологічне ядро розроблюваної системи.

3.1.1. Реалізація алгоритму багаторівневих пріоритетних черг та обробка даних

Серцем системи є алгоритм керування транспортними потоками на основі динамічних пріоритетів [2]. Оскільки матеріальні потоки у гнучких виробничих системах можна формалізувати як мережі масового обслуговування, виникає потреба у високоефективному сортуванні елементів у реальному часі [5, 6]. Стандартні списки Python (**list**) мають часову складність $O(n \cdot \log n)$ при регулярному пересортуванні, що є неефективним для великої інтенсивності матеріального потоку.

Для оптимізації операцій додавання та вилучення деталей з урахуванням їхніх пріоритетів було обрано вбудовану бібліотеку **heapq** [7]. Цей модуль реалізує алгоритм **бінарної купи (binary heap)**, що забезпечує виконання операцій додавання елемента (**heappush**) та вилучення елемента з найвищим пріоритетом (**heappop**) за логарифмічний час $O(n \cdot \log n)$ [7]. Це гарантує стабільну швидкість роботи системи навіть при значному перевантаженні буферів зберігання. При реалізації враховано особливості стабільності сортування кортежів: черга оперує структурами типу (**priority, insert_order, item**), що запобігає конфліктам порівняння об'єктів при однакових пріоритетах [7].

Для первинного аналізу, структурування масивів вхідних даних та ведення журналів обліку деталей використано бібліотеку **pandas** [8]. Застосування об'єктів **DataFrame** та **Series** дозволяє виконувати швидку фільтрацію, групування та агрегування статистичних показників завантаження дільниць [8]. Модуль **pandas** забезпечує зручну інтеграцію з локальними сховищами даних та дозволяє ефективно трансформувати лог-файли для подальшого аналізу ефективності логістичного ланцюга [8, 1].

3.1.2. Генерація випадкових подій та імітаційне моделювання

Оскільки реальні виробничі процеси характеризуються стохастичністю (випадковий час надходження заготовок на лінію, коливання часу виконання технологічних операцій), для тестування системи необхідний модуль імітаційного моделювання [5].

Для генерації випадкових величин та моделювання законів розподілу ймовірностей (зокрема, експоненціального розподілу для вхідного потоку заявок та нормального розподілу для часу транспортування візками) використано вбудований модуль **random**, а також бібліотеку **NumPy** [5, 9]. Використання **NumPy** дозволяє генерувати великі вектори випадкових подій за допомогою оптимізованих C-функцій, що значно прискорює проходження імітаційних тактів (**ticks**) системи [9].

3.1.3. Створення інформаційного забезпечення системи

Для забезпечення надійної фіксації стану системи, збереження специфікацій моделей (наприклад, для моделей FPV-дронів Mark4-Light, Standard, Pro) та ведення повної історії переміщень деталей інтегровано систему керування базами даних (СКБД) **sqlite3** [3].

Вибір **sqlite3** обґрунтований тим, що вона є вбудованою безсерверною базою даних, яка зберігає всю інформацію в одному локальному файлі. Це усуває потребу в розгортанні важких серверних рішень (як-от PostgreSQL або MySQL) безпосередньо в цеху, спрощує мобільність програмного забезпечення та гарантує повну підтримку принципів **ACID (Atomicity, Consistency, Isolation, Durability)** для запобігання втрати даних при збоях [3]. Структура реляційних таблиць містить зв'язки «багато до багатьох» для специфікацій уніфікованих деталей та індекси для швидкого пошуку полів **calculated_priority** та **timestamp**.

3.1.4. Проєктування графічного інтерфейсу користувача (GUI)

Для забезпечення оперативної взаємодії оператора з системою розподілу деталей було спроєктовано графічний інтерфейс користувача. Під час вибору технологій аналізувалися варіанти використання комерційної бібліотеки **PySide6 (Qt)** та стандартної бібліотеки **tkinter**.

Для розробки, первинного тестування та впровадження прототипу системи керування пріоритетними чергами перевагу було віддано бібліотеці **tkinter**. Головними перевагами **tkinter** є:

- Входження до стандартної збірки Python, що виключає необхідність встановлення додаткових зовнішніх залежностей при розгортанні системи на виробничих обчислювальних станціях;
- Низьке споживання системних ресурсів (легковажність) та висока швидкість відгуку вікон графічного контуру;
- Максимальна мобільність коду при перенесенні між різними операційними системами (Windows/Linux), які використовуються на промислових контролерах.

- Графічний інтерфейс включає чотири функціональні вкладки («Склад», «Дільниці», «Збірки», «Журнал»), що забезпечує всебічний візуальний моніторинг контуру управління матеріальними потоками у режимі реального часу.

3.2. Розробка програмного забезпечення системи

Розроблене програмне забезпечення призначене для автоматизованого керування потоками деталей у багаторівневій транспортній системі з використанням механізму динамічних пріоритетів. Програмна система побудована за модульним принципом, що забезпечує розподіл функцій між окремими компонентами та спрощує подальше розширення функціональних можливостей.

Основним керуючим модулем системи є файл **main.py** (Додаток Б), який виконує ініціалізацію програмного середовища, підключення до бази даних та запуск графічного інтерфейсу користувача. Після запуску система отримує доступ до локальної бази даних **SQLite** [3], у якій зберігається інформація про складські запаси деталей, виробничі дільниці, виробничий план, результати складання виробів та журнал подій.

Функціонування прикладного програмного забезпечення системи здійснюється відповідно до алгоритму, наведеного на рисунку 3.1.

Рис. 3.1. Функціональна схема прикладного програмного забезпечення

Центральним елементом програмної логіки є модуль **ProductionSimulator** (Додаток Б), який реалізує імітаційне моделювання виробничого процесу. Під час роботи модуль послідовно виконує операції вибору деталей зі складу, контролю якості, визначення маршруту переміщення та формування готових виробів. Кожна виконана дія фіксується в журналі подій, що дозволяє відстежувати стан системи в режимі реального часу.

Після вибору чергової деталі зі складу виконується процедура контролю якості. Залежно від результатів перевірки деталь може бути допущена до подальшого використання, направлена на доопрацювання, переробку або перенаправлена для використання у виробах з нижчим рівнем пріоритету. Такий підхід дозволяє підвищити ефективність використання ресурсів та зменшити кількість виробничих втрат.

Для визначення напрямку переміщення деталей використовується модуль **PriorityEngine** (Додаток Б), який реалізує механізм багаторівневих пріоритетних черг. При розрахунку пріоритету враховуються потреби виробничих дільниць у конкретних деталях, важливість компонентів для складання виробів та поточний стан виконання виробничого плану. Застосування пріоритетного обслуговування дозволяє забезпечити більш раціональний розподіл ресурсів транспортної системи та скоротити час очікування критично важливих компонентів [1, 5].

Після визначення цільової дільниці деталі передаються до відповідних буферів накопичення. Система автоматично контролює наявність необхідного комплекту компонентів для кожного виробу. У випадку достатньої кількості деталей запускається процедура складання виробу, після чого результати виконання операції зберігаються у базі даних та відображаються в інтерфейсі користувача.

Обмін інформацією між усіма програмними модулями здійснюється через базу даних **SQLite** [3]. Використання єдиного сховища даних забезпечує цілісність інформації про переміщення деталей, виконання виробничих операцій та поточний стан складських запасів. Крім того, накопичення статистичних даних

дозволяє виконувати подальший аналіз ефективності логістичних процесів та роботи виробничої системи загалом [1].

Для взаємодії оператора із системою використовується графічний інтерфейс, реалізований засобами бібліотеки **tkinter** [7]. Інтерфейс забезпечує введення виробничого плану, відображення стану складу, моніторинг виробничих дільниць, перегляд журналу подій та контроль результатів складання виробів. Завдяки цьому користувач отримує можливість оперативно контролювати роботу системи та аналізувати результати її функціонування.

Таким чином, розроблене програмне забезпечення забезпечує автоматизоване керування потоками деталей на основі механізму пріоритетних черг, дозволяє моделювати виробничі процеси та здійснювати моніторинг усіх етапів переміщення і складання деталей у багаторівневій транспортній системі.

Для наочного відображення взаємодії основних програмних модулів розробленої системи побудовано структурну схему програмного забезпечення, наведену на рисунку 3.2.

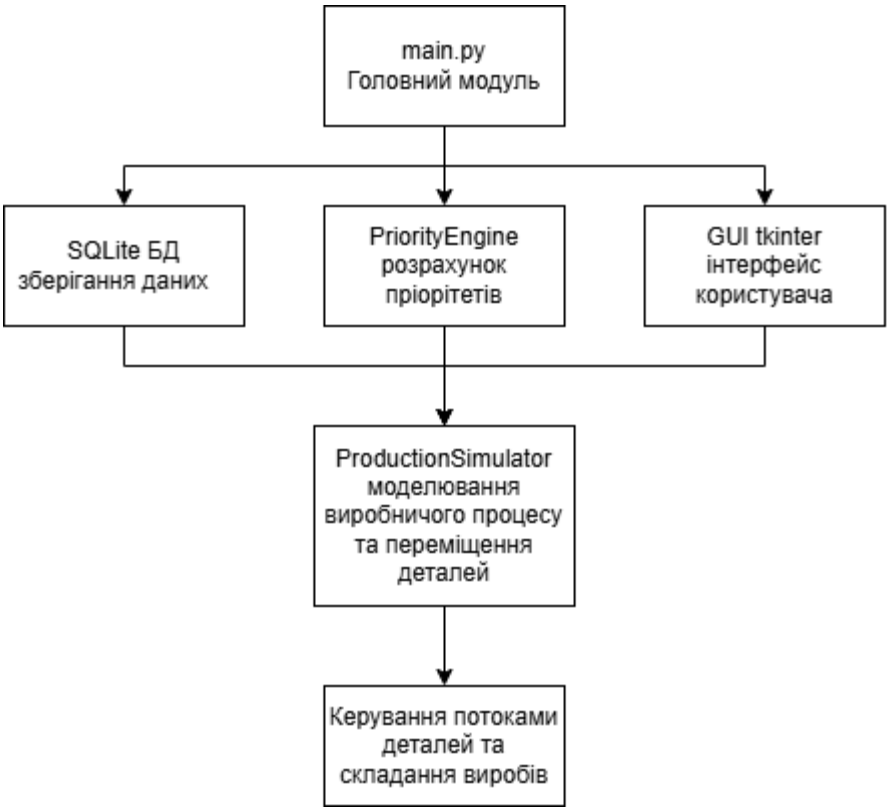


Рис 3.2. Структурна схема програмного забезпечення

3.3. Методичне забезпечення системи

Для функціонування системи автоматизованого контрою потоків деталей розроблено інтерфейс користувача.

3.3.1. Головне вікно програми

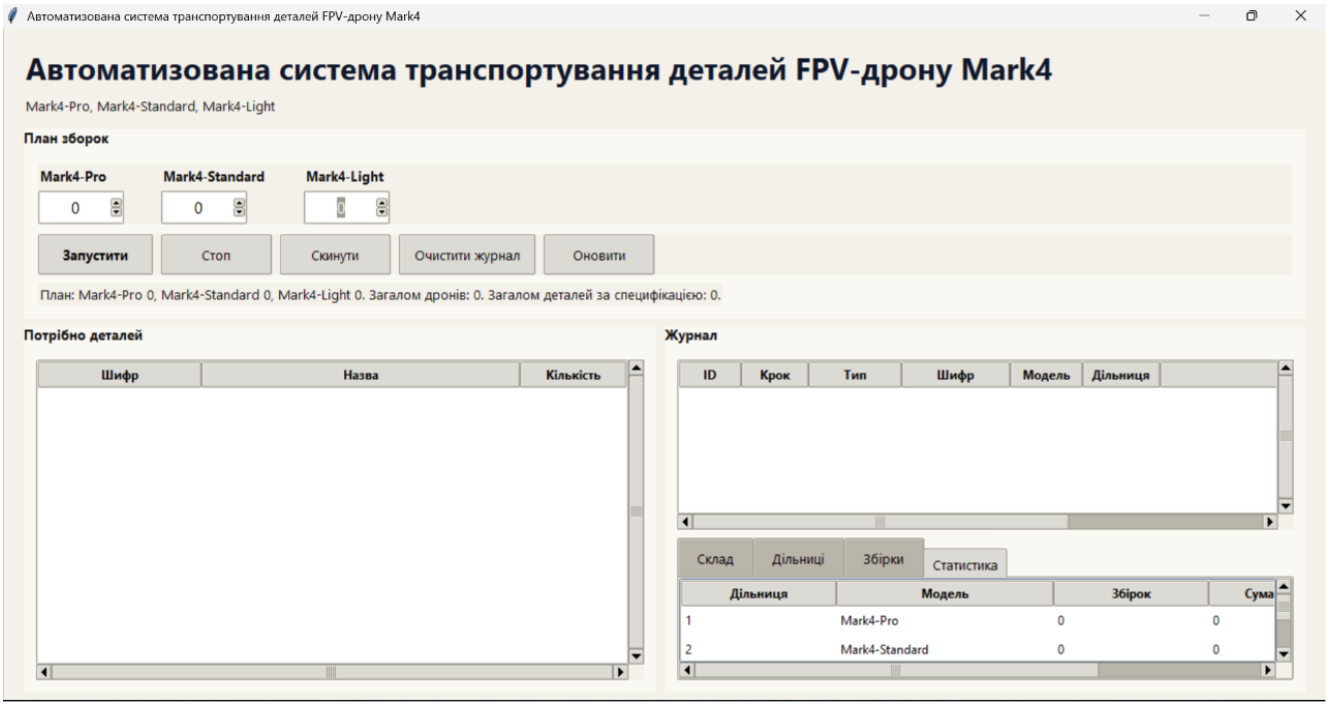


Рис. 3.3. Головне вікно програмного забезпечення призначене для взаємодії оператора з системою керування потоками деталей.

У верхній частині вікна (на рис. 3.3) розташована панель формування виробничого плану, де користувач задає кількість виробів кожної моделі FPV-дронів. Після введення даних система автоматично виконує розрахунок необхідної кількості комплектуючих та відображає результати у відповідних таблицях.

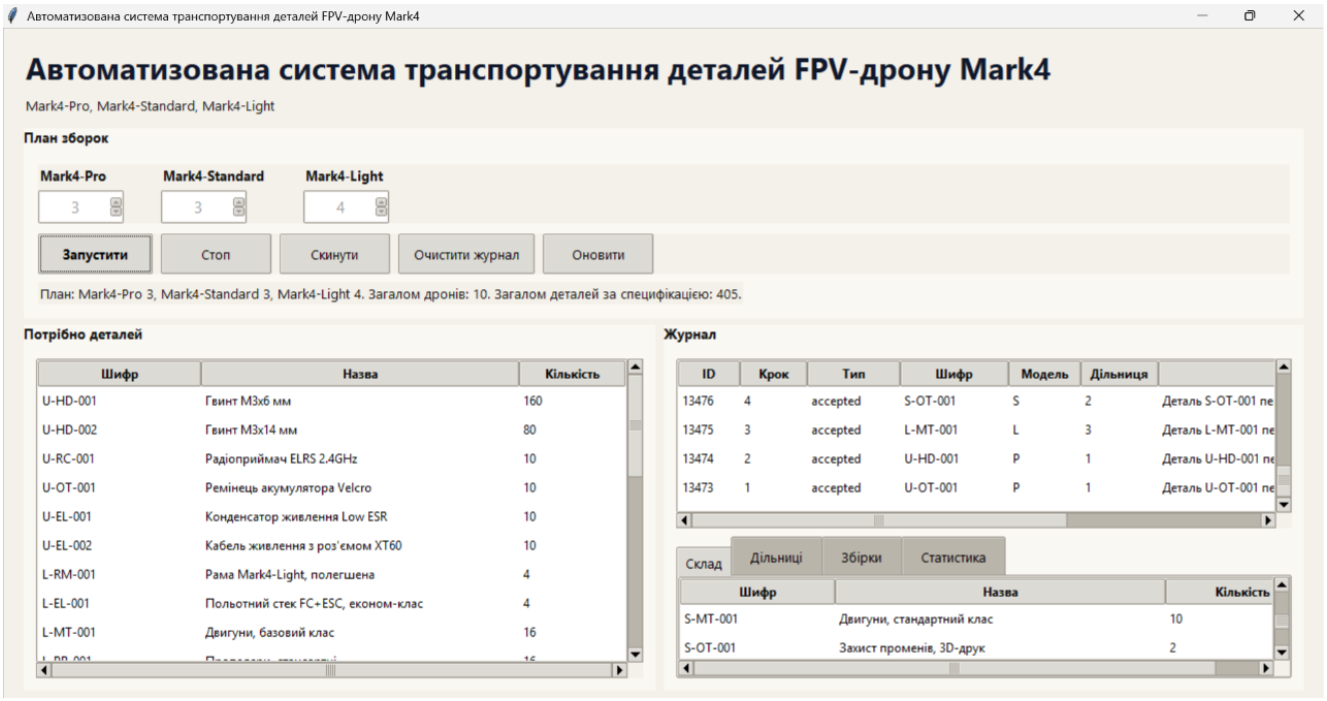


Рис. 3.4. Головне вікно під час роботи

У центральній частині вікна знаходяться інформаційні панелі, які відображають поточний стан складу, виробничих буферів, журнал подій та статистичні дані моделювання. Така структура інтерфейсу забезпечує зручний контроль усіх основних параметрів виробничого процесу в режимі реального часу (на рис. 3.4).

3.3.2. Формування виробничого плану

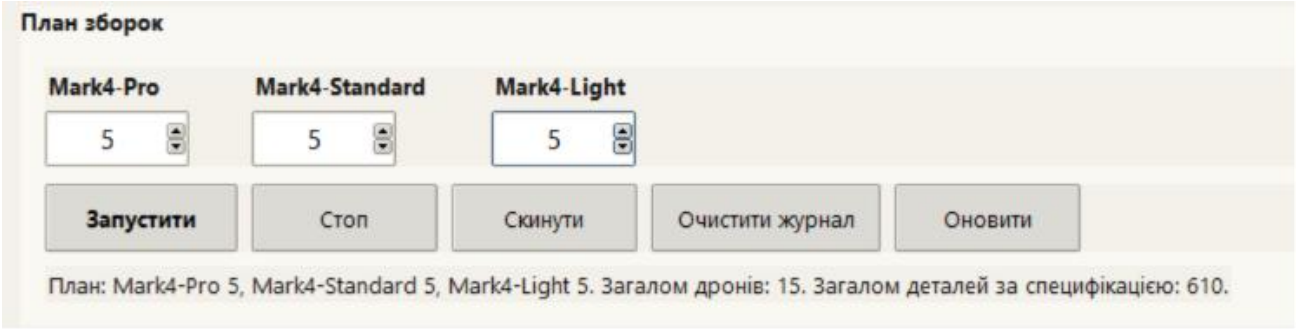


Рис. 3.5. Вводити кількість деталей, формування виробничого плану

Дане вікно на рисунку 3.5 використовується для задання кількості виробів, які необхідно виготовити в межах поточного виробничого циклу. Для кожної моделі дрона передбачене окреме поле введення, що дозволяє формувати план довільної структури залежно від виробничих потреб.

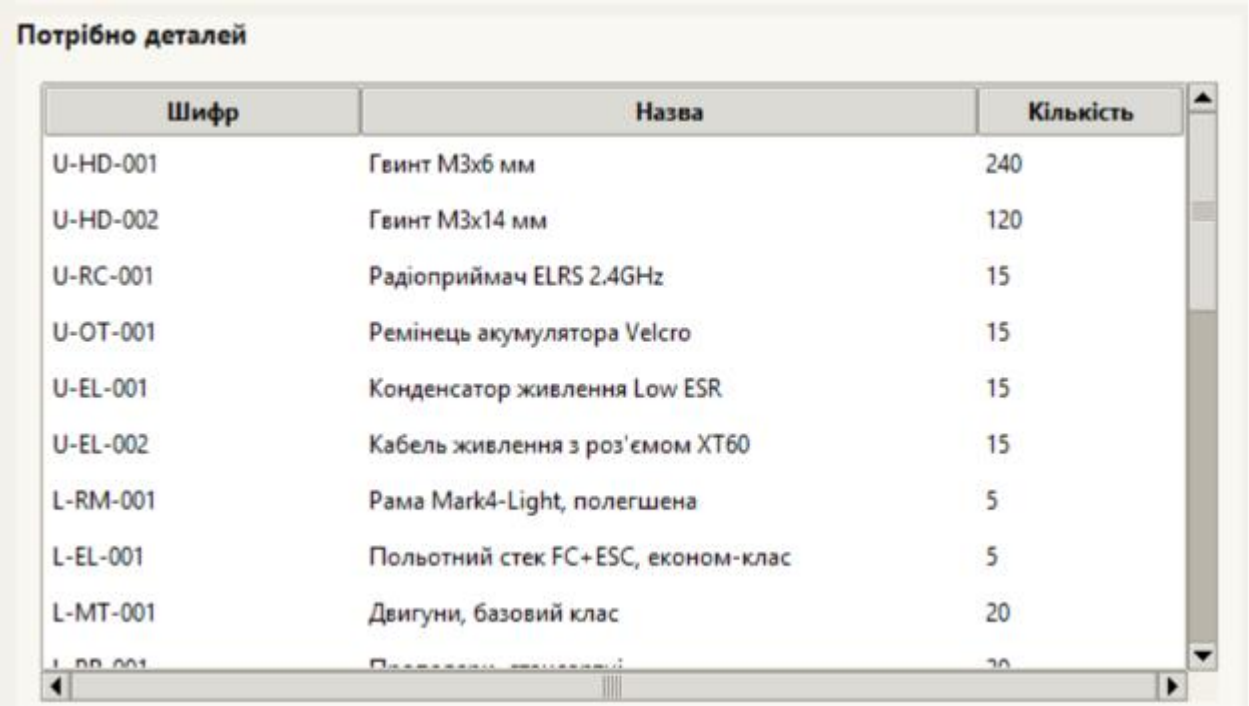


Рис. 3.6. Автоматично підрахована кількість деталей

Після зміни значень система автоматично розраховує загальну кількість виробів та потребу в комплектуючих (на рис. 3.6). Отримана інформація використовується для подальшого формування пріоритетних черг і виконання імітаційного моделювання.

3.3.3. Моніторинг складських запасів та виробничих дільниць

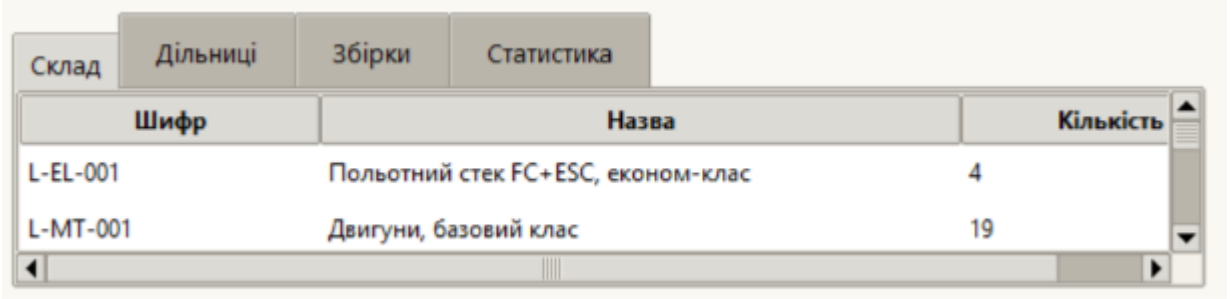


Рис. 3.7. Вкладка склад

На даній вкладці на рисунку 3.7 представлена інформація про поточний залишок деталей на складі. Дані оновлюються автоматично в процесі моделювання після кожної операції переміщення або використання компонентів.

Відображення актуального стану складу дозволяє контролювати витрати комплектуючих та своєчасно виявляти потенційний дефіцит ресурсів.

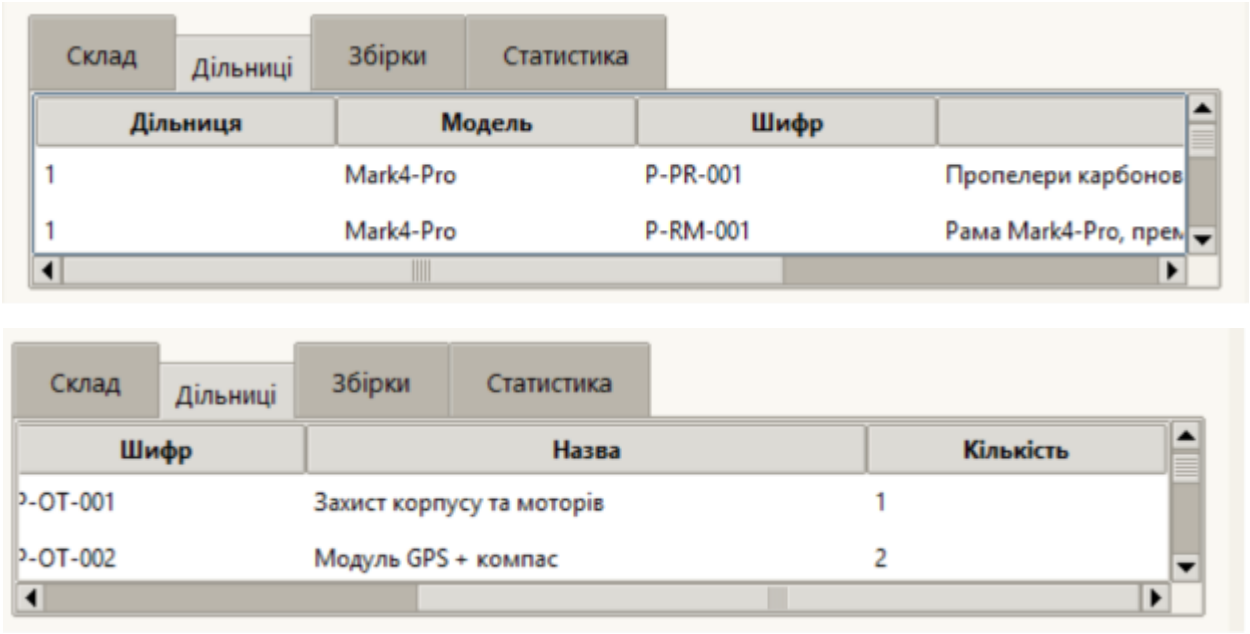


Рис. 3.8. Список деталей для дільниць у вікні та кількість прибувших на дільницю.

У даному вікні на рисунку 3.8 відображаються запаси деталей, що знаходяться безпосередньо у виробничих буферах окремих дільниць. Інформація використовується алгоритмом багаторівневих пріоритетів для визначення необхідності транспортування нових компонентів.

Аналіз стану буферів дозволяє оцінювати рівень забезпеченості виробничих дільниць та ефективність роботи транспортної системи.

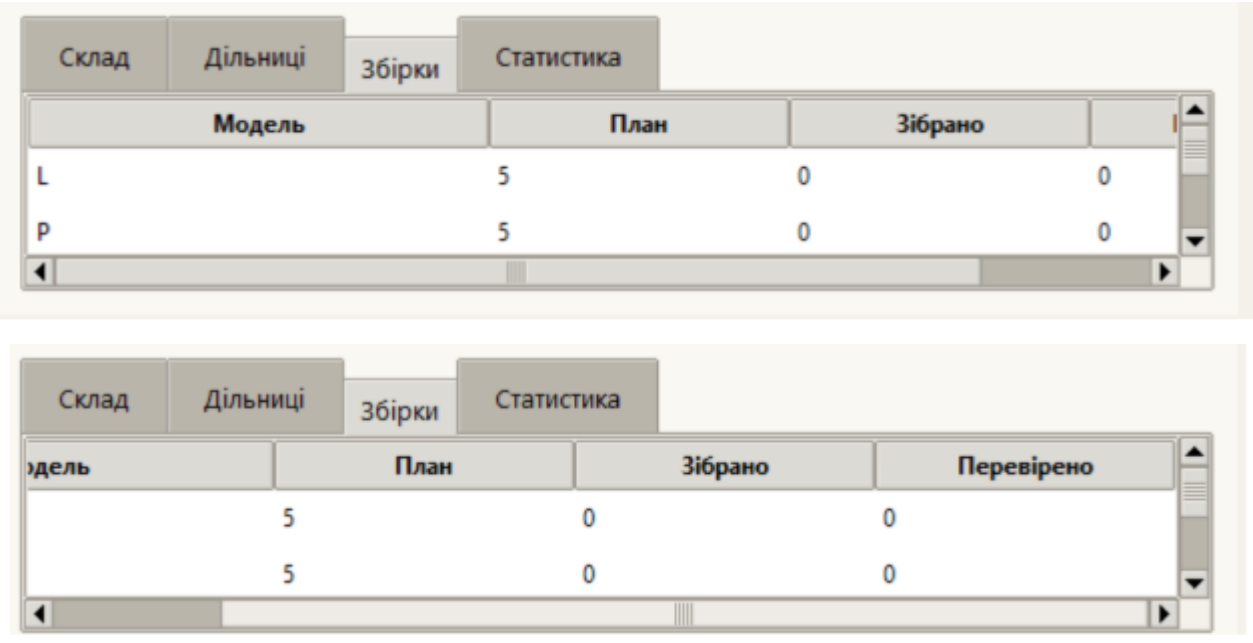


Рис. 3.9. Автоматичне оновлення кількості зроблених виробів

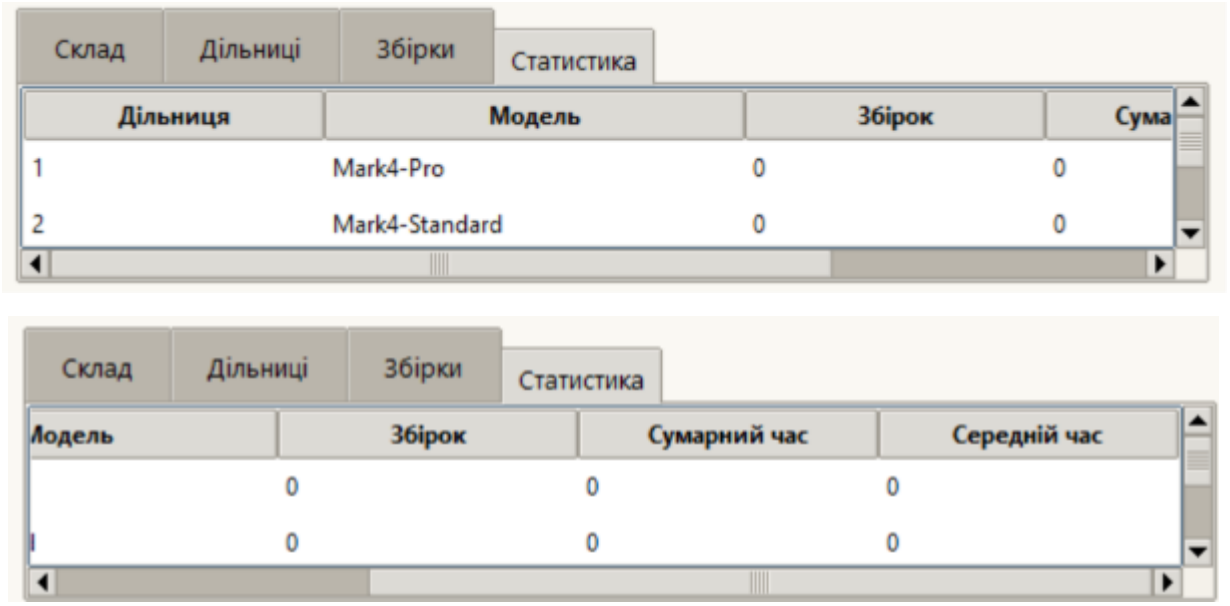


Рис. 3.10. Підрахунок скільки часу займає збірка кожної з моделей

У цих вікнах (на рис. 3.9-3.10) ми можемо побачити кількість деталей на даний момент часу на різних точках транспортної системи.

У таблицях 3.1.-3.4 відображається перелік деталей, необхідних для виконання сформованого виробничого плану. Для кожної позиції вказуються її шифр, найменування та розрахована кількість.

Автоматичне формування даної таблиці дозволяє швидко оцінити потребу в ресурсах та перевірити забезпеченість виробництва комплектуючими до початку моделювання.

Таблиця 3.1. Універсальні деталі

Універсальні деталі (код моделі: U)		
Шифр	Найменування деталі	Кількість
U-HD-001	Гвинт M3x6 мм	по 4 на кожну модель
U-HD-002	Гвинт M3x14 мм	по 2 на кожну модель
U-RC-001	Радіоприймач ELRS 2.4GHz	по 1 на кожну модель
U-OT-001	Ремінець акумулятора Velcro	по 1 на кожну модель

U-EL-001	Конденсатор живлення Low ESR	по 1 на кожну модель
U-EL-002	Кабель живлення з роз'ємом XT60	по 1 на кожну модель

Таблиця 3.2. Mark4-Light

Mark4-Light (код: L)		
Шифр	Найменування деталі	Кількість штук
L-RM-001	Рама Mark4-Light (полегшена)	1
L-EL-001	Польотний стек (FC+ESC) – економ-клас	1
L-MT-001	Двигуни – базовий клас	4
L-PR-001	Пропелери – стандартні	4
L-VC-001	Відеосистема – аналог (базова)	1

Таблиця 3.3. Mark4-Standard

Mark4-Standard (код: S)		
Шифр	Найменування деталі	Кількість штук
S-RM-001	Рама Mark4 – Standard	1
S-EL-001	Польотний контролер (FC) – стандарт	1
S-EL-002	Регулятор швидкості (ESC) – стандарт	1

S-MT-001	Двигуни – стандартний клас	4
S-PR-001	Пропелери – посилені	4
S-VC-001	Відеосистема – аналог (середній клас)	1
S-OT-001	Захист променів (3D-друк)	2

Таблиця 3.4. Mark4-Pro

Mark4-Pro (код: P)		
Шифр	Найменування деталі	Кількість штук
P-RM-001	Рама Mark4-Pro (преміум)	1
P-EL-001	Польотний контролер – преміум	1
P-EL-002	Регулятор швидкості (ESC) – преміум	1
P-MT-001	Двигуни – преміум (низька вібрація)	4
P-PR-001	Пропелери – карбонові	4
P-VC-001	Відеосистема – цифрова HD	1
P-OT-001	Захист корпусу та моторів (комплект)	2
P-OT-002	Модуль GPS + компас	1

3.3.4. Відображення процесу виконання виробничого плану

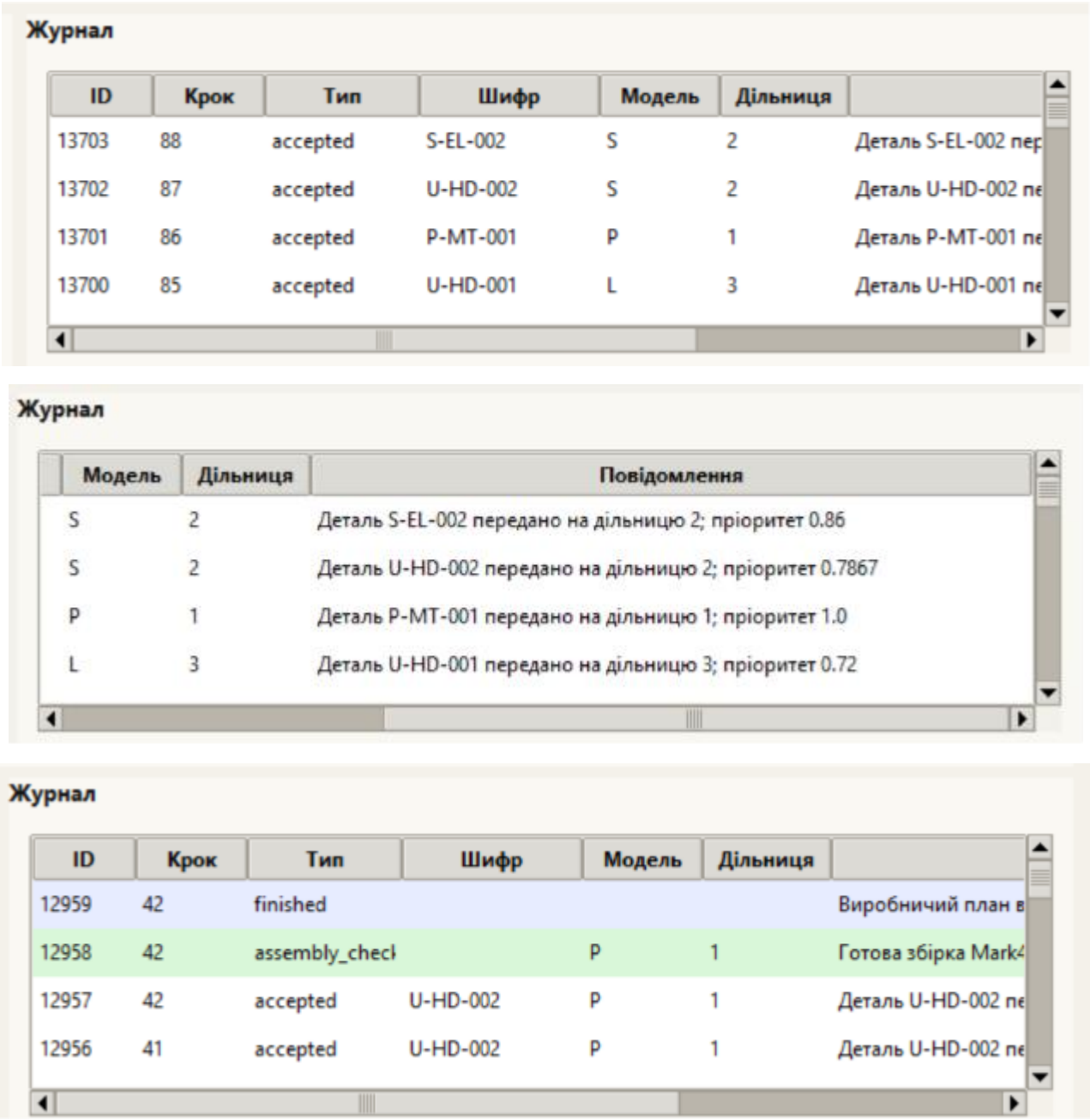


Рис. 3.11. Журнал подій

Журнал подій (на рис. 3.11) призначений для реєстрації всіх основних операцій, що виконуються під час роботи програми. У ньому відображаються повідомлення про транспортування деталей, проходження контролю якості, перенаправлення компонентів між дільницями, а також виникнення дефіциту ресурсів.

Використання журналу дозволяє відстежувати послідовність виконання виробничих операцій та аналізувати причини можливих відхилень від запланованого режиму роботи системи.

Під час виконання виробничого плану у журнал в режимі реального часу виводяться результати, кожна готова збірка виділяється зеленим, а кінець виробничого плану синім.

3.3.5. Аналіз результатів роботи системи

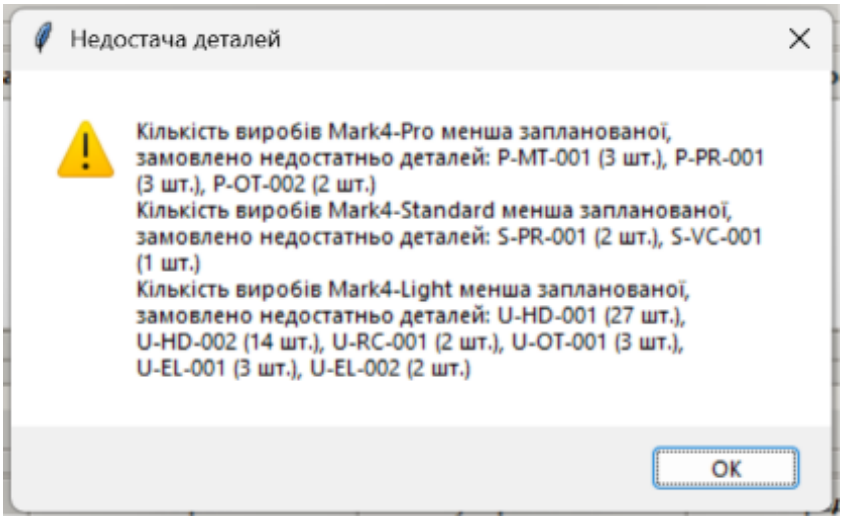


Рис. 3.12. Після виконання всього виробничого плану з'являється повідомлення з підрахунком недостач через брак.

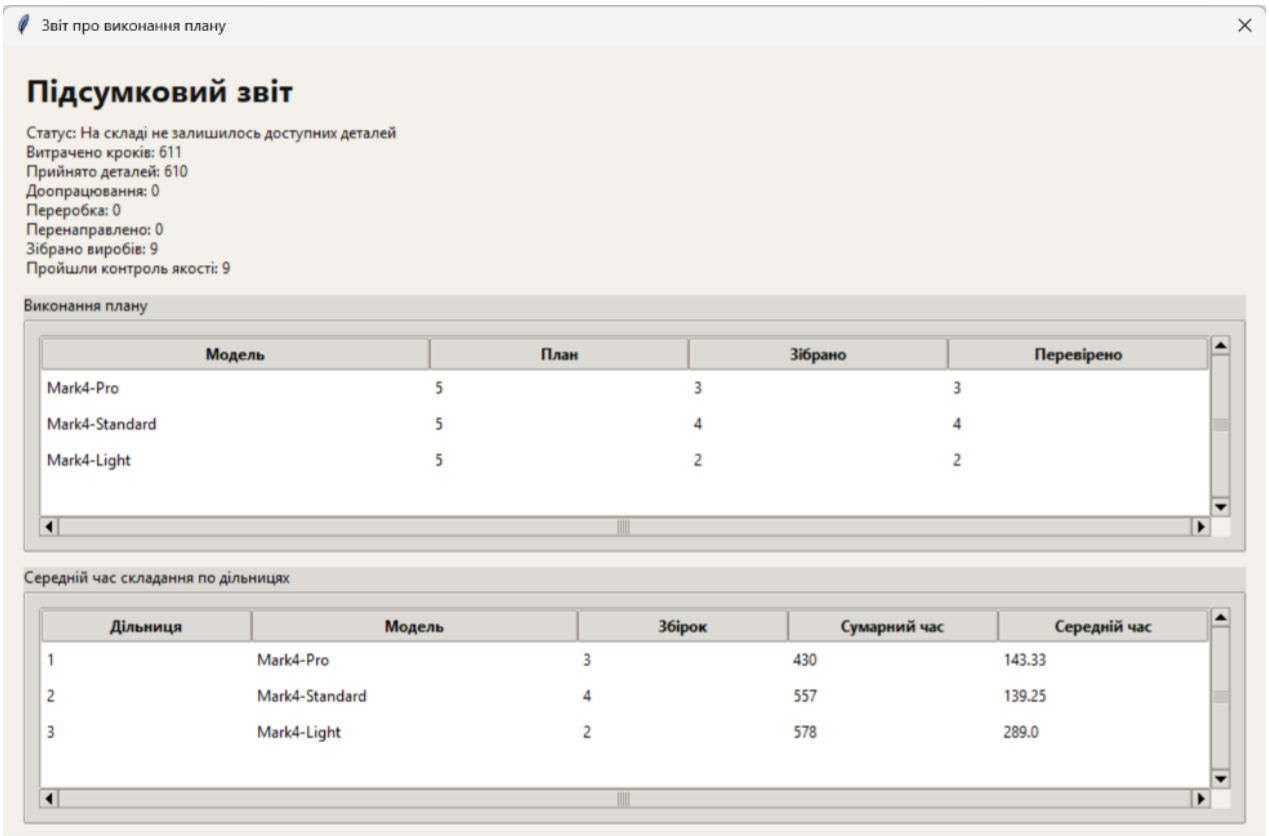


Рис. 3.13. Підсумковий звіт

Після завершення імітаційного моделювання система автоматично формує підсумковий звіт (на рис. 3.13) та повідомлення щодо недостачі (на рис. 3.14). У звіті відображаються показники виконання виробничого плану, кількість оброблених деталей, результати контролю якості та статистика складання виробів.

Отримані дані дозволяють оцінити ефективність роботи алгоритму керування потоками деталей та визначити можливі напрями подальшого вдосконалення системи.

3.4. Практична реалізація автоматизованої системи з використанням багаторівневих пріоритетів

3.4.1. Формування виробничого плану та визначення потреби в деталях

Будь-яке автоматизоване виробництво починається з чіткого планування, оскільки без розуміння кінцевої мети неможливо правильно організувати логістику всередині цеху [1]. У розробленій керуючій програмі цей процес є початковою точкою запуску імітаційного моделювання. Користувач через графічний інтерфейс має можливість самостійно задавати обсяги випуску виробів, орієнтуючись на три основні лінійки продукції підприємства: **преміальні моделі Mark4-Pro, стандартні Mark4-Standard та базові Mark4-Light.**

З погляду програмної логіки, функція зчитування плану, яка детально описана в модулі **gui.py** (Додаток Б), збирає введені користувачем числові значення з текстових полів інтерфейсу. Ці дані передаються до об'єкта симулятора, де за допомогою спеціального методу відбувається фіксація поточного замовлення та підготовка внутрішнього середовища. Оскільки для кожного типу виробів передбачена своя специфікація та структура матеріальних витрат, звичайного збереження загальної кількості одиниць недостатньо — системі необхідно миттєво оцінити реальну потребу в комплектуючих елементах для всього ланцюга постачань [1, 3].

Для розрахунку сумарної кількості необхідних вузлів у програмі реалізовано алгоритм автоматичного калькулювання матеріальних потреб, який функціонує в симуляторі виробничих процесів (модуль **simulator.py** у Додатку Б). Механізм

обчислення побудовано на динамічному зв'язку між планом випуску та базою даних, де зберігаються технічні паспорти кожної деталі (за це відповідає клас ініціалізації сховища даних, наведений у модулі **database.py** у Додатку Б). Коли користувач підтверджує план, програма запускає прихований цикл, який покроково перемножує кількість запланованих безпілотних апаратів кожної моделі на нормативні витрати відповідних компонентів.

У процесі розрахунку потреби система враховує, що різні моделі виробів можуть містити як унікальні деталі, так і взаємозамінні вузли. Наприклад, рами преміум-класу підходять лише для моделей Mark4-Pro, тоді як базові пропелери чи двигуни можуть використовуватись у кількох конфігураціях одночасно [2]. Керуюча програма сумує всі ці показники та формує єдину підсумкову матрицю матеріальних потреб. Отриманий результат виводиться на екран оператора у вигляді наочної інформаційної довідки або оновленого рядка стану, що дозволяє візуально оцінити масштаб майбутнього виробничого циклу ще до моменту фізичного запуску конвеєра.

Особливістю цього етапу є те, що визначена потреба в деталях не є статичною. Оскільки в реальних умовах автоматизованого середньосерійного виробництва часто виникають затримки постачань, випадковий брак деталей або збої у роботі обладнання, сформований план має гнучко підлаштовуватися під поточну ситуацію [2]. Розраховані на початку роботи значення потреби передаються далі за кодом — до функцій розподілу та пріоритезації в модулі **priority.py** (Додаток Б), де вони безпосередньо впливають на динамічну зміну коефіцієнтів у чергах на конвеєрних лініях. Таким чином, початкове формування плану забезпечує систему базовими орієнтирами, що дозволяє їй ефективно балансувати між завантаженням складальних дільниць та наявними складськими запасами компонентів [3].

3.4.2. Формування пріоритетів та керування потоками деталей

Основою гнучкої системи керування потоками є перехід від статичних правил черги FIFO «перший прийшов — перший обслужений» до динамічного

розподілу ресурсів [5]. У сучасних умовах автоматизованого виробництва важливо забезпечити баланс між завантаженістю обладнання та терміновістю виконання окремих замовлень, оскільки класичні підходи до організації складів часто не враховують стохастичну природу виробничих процесів [2]. Для реалізації цього механізму в розробленій системі було створено клас **PriorityEngine**, який знаходиться у модулі **priority.py** (Додаток Б).

Цей клас виконує ключову роль у прийнятті рішень щодо черговості обробки. Замість використання жорстко заданих значень, **PriorityEngine** проводить динамічну оцінку, що базується на параметрах об’єкта **PriorityWeights** (Додаток Б). Цей підхід дозволяє коригувати стратегію обробки залежно від поточного стану виробничих запасів та вимог виробничого плану [1]. Коефіцієнти, що визначаються в **PriorityWeights**, дозволяють гнучко налаштовувати вагу таких факторів, як критичність нестачі деталі та її приналежність до певної виробничої ділянки.

Процес керування потоками тісно інтегрований з симулятором виробництва, логіка якого реалізована в модулі **simulator.py** (Додаток Б). Взаємодія полягає в тому, що після кожного кроку моделювання система перевіряє рівень запасів. Механізм розрахунку пріоритету використовує відношення поточної нестачі до цільового рівня буфера. Це дозволяє системі автоматично підвищувати пріоритет тих деталей, які є критично необхідними для виконання поточного плану, що суттєво зменшує ймовірність простоїв на окремих етапах конвеєра [3].

Така модель керування відповідає концепції багаторівневих систем обслуговування, де черга не є статичною, а постійно перераховується залежно від імовірнісних характеристик надходження вимог [6]. Застосування подібних багатокритеріальних підходів до визначення пріоритетів дає можливість досягти ефективнішого розподілу деталей між робочими станціями, уникаючи надмірного накопичення незавершеного виробництва [3]. Завдяки такій інтеграції, система здатна ефективно працювати навіть за умови виникнення затримок постачань або випадкових збоїв у роботі обладнання [2, 4].

Отже, використання класу **PriorityEngine** у поєднанні з логікою, закладеною в simulator.py, забезпечує адаптивність системи до змін у реальному виробничому середовищі. Така архітектура програмного забезпечення мінімізує вплив людського фактора на прийняття оперативних рішень щодо черговості обробки деталей, забезпечуючи при цьому оптимізацію використання транспортних засобів [2].

3.4.3. Розподіл деталей між виробничими дільницями

Ефективний розподіл деталей у системі є завершальним етапом алгоритму керування потоками, що дозволяє реалізувати прийняті раніше пріоритетні рішення на фізичному рівні [6]. Основне завдання цього етапу полягає в тому, щоб забезпечити гнучкість виробничої системи та мінімізувати простої обладнання, навіть за умов нестабільності вхідних потоків [2].

Архітектурно розподіл базується на динамічному аналізі стану кожного окремого вузла системи. Для забезпечення високої швидкодії при опрацюванні великих масивів даних про стан буферів та черг, у програмному забезпеченні застосовуються інструменти бібліотеки **pandas** [8], які дозволяють ефективно групувати та фільтрувати дані про наявні заготовки. Це дає можливість системі в режимі реального часу «бачити» завантаженість дільниць та приймати зважені рішення щодо маршрутизації кожної деталі [1].

Логіка розподілу передбачає багаторівневу перевірку умов. Насамперед, програмний модуль здійснює запит до бази даних для отримання актуальних параметрів цільової дільниці. Важливим аспектом тут є використання чисельних методів оцінки, що реалізуються за допомогою бібліотеки **NumPy** [9], для порівняння поточного завантаження дільниць з гранично допустимими показниками. Якщо основна дільниця, для якої призначена деталь, має вільний ресурс, система ініціює процедуру передачі. У випадках, коли дільниця виявляється зайнятою або деталь не відповідає технічним вимогам конкретного робочого місця, система автоматично переходить до процедури перенаправлення.

Для обробки таких виняткових ситуацій у коді передбачено функцію **_redirect_lower_model** (Додаток Б). Її завданням є пошук альтернативного маршруту або місця тимчасового зберігання, що мінімізує збитки від виникнення черг, оскільки «мінімізація збитків від існування черги» є фундаментальним завданням дослідника технічної системи [5]. Така архітектура відповідає принципам багатокритеріальної оптимізації, де балансування навантаження між різними зонами складу чи виробництва дозволяє уникнути перевантаження окремих вузлів [3].

Окремо варто зазначити, що процес логування подій передачі деталі відбувається в той самий момент, коли система приймає рішення про розподіл. Це забезпечує актуальність стану виробничого середовища та дозволяє отримати звіт про місцезнаходження будь-якої заготовки в системі в будь-який момент часу. Як зазначається у дослідженнях, саме безперервна інтеграція інформаційних потоків з фізичними є критичною для досягнення цільових показників ефективності (KPI) та стабільної роботи ланцюга постачань [1].

Завдяки використанню структур даних, оптимізованих для швидкого доступу до пріоритетних елементів [7], процес розподілу не створює додаткових затримок у виробничому циклі. Це значно скорочує час виконання замовлення (lead time), що є одним із ключових параметрів при оцінці ефективності автоматизованих виробничих систем [4]. Таким чином, розподіл між дільницями стає не просто технічною операцією передачі об'єкта, а інтелектуальним процесом, що дозволяє адаптуватися до змінних умов середньосерійного виробництва [2].

3.4.4. Обробка нестандартних виробничих ситуацій

Функціонування будь-якої автоматизованої системи керування транспортними потоками неминуче стикається з умовами, що виходять за межі стандартного алгоритму. У реальних виробничих середовищах часто виникають випадки, коли деталь не відповідає технічним вимогам конкретної дільниці або коли буферні зони заповнені понад встановлені ліміти. Ігнорування таких ситуацій

може призвести до блокування всього транспортного потоку, тому розроблена система передбачає механізми адаптивної обробки виняткових подій.

Базовим елементом цієї логіки є процедура перенаправлення заготовок, реалізована через функцію **_redirect_lower_model** (див. Додаток Б). Коли стандартний алгоритм розподілу, що базується на пріоритетах, не може підібрати вільний слот або припустимий маршрут, система автоматично ініціює виклик цього методу. Основне завдання такої архітектури полягає у пошуку альтернативних шляхів — наприклад, тимчасового зберігання деталі в буфері або перенаправлення на спеціалізовані модулі для додаткової обробки. Як слушно зауважують дослідники теорії систем масового обслуговування, завдання дослідника технічної системи часто полягає саме у тому, щоб мінімізувати збитки від існування черги, розглядаючи її як «неминуче зло» [5].

Процес прийняття рішень про перенаправлення тісно інтегрований з модулем логування подій. У момент, коли система стикається з відхиленням, вона через методи роботи з базою даних фіксує поточний стан середовища: тип деталі, код дільниці, причину відмови та час прийняття рішення. Така прозорість дій системи є критично важливою, адже безперервна інтеграція інформаційних потоків з фізичними рухами деталей дозволяє досягати стабільних показників ефективності (KPI) та вчасно реагувати на збої [1].

З точки зору багатокритеріальної оптимізації, такий підхід дозволяє гнучко адаптувати стратегії управління відповідно до бізнес-пріоритетів. Якщо система бачить, що навантаження на ключові вузли стає критичним, вона балансує потік, уникаючи перевантаження окремих ділянок, що є типовою проблемою складних виробничих систем [3]. У випадках, коли виникає затор, алгоритм моделювання черг повинен враховувати взаємозалежність вузлів та динамічну зміну їхніх станів, щоб уникнути колапсу системи [6].

Технічно це реалізовано за допомогою вбудованих методів роботи зі структурами даних та словниками, що дозволяє програмі практично миттєво оцінити стан «черги» і прийняти рішення без значних затримок. Оскільки динамічна зміна щільності потоків заготовок вимагає миттєвого перерахунку,

використання оптимізованих алгоритмів обробки даних є необхідною умовою для підтримки стабільності [7]. Це дозволяє системі не просто «рятувати» деталь від простою, а й забезпечити цілісність логістичного процесу навіть за умов високої завантаженості виробничих потужностей.

Таким чином, розроблений механізм обробки нестандартних ситуацій перетворює систему з жорсткого алгоритму на гнучкий інструмент, здатний до саморегуляції. Це мінімізує вплив людського фактора та випадкових збоїв, що є ключовим для сучасних систем автоматизації [2].

3.4.5. Імітаційне моделювання виробничого процесу

Заключним етапом програмної реалізації є імітаційне моделювання, яке дозволяє перевірити працездатність розробленої логіки в умовах, максимально наближених до реального виробництва. Для цього в керуючій програмі було створено клас **ProductionSimulator** (Додаток Б), основна функція якого полягає у відтворенні циклів роботи системи протягом заданого часового проміжку.

Процес симуляції базується на принципах теорії масового обслуговування, де черга розглядається як інструмент балансування навантаження, що дозволяє мінімізувати збитки від простоїв [5]. Використання моделі дозволяє оцінити, як саме розроблений алгоритм багаторівневих пріоритетів впливає на загальну продуктивність, враховуючи специфіку гнучких виробничих систем [6].

Під час запуску імітації (метод **run** класу **ProductionSimulator**, Додаток Б), система автоматично звертається до модуля бази даних database.py. Кожен крок імітації супроводжується ініціалізацією параметрів потоку заявок та перевіркою стану складських буферів. Це критично важливо, адже саме динамічна зміна щільності потоків деталей вимагає від системи миттєвої реакції для уникнення виникнення «вузьких місць» у системі постачання [1].

Окремої уваги заслуговує механізм формування звітності після завершення циклу моделювання. Завдяки інтеграції симулятора з базою даних, наприкінці роботи алгоритму ми отримуємо розгорнуту статистику: загальну кількість прийнятих деталей, обсяги необхідної переробки (**rework**), відсоток використання

вторинної сировини (**recycle**) та кількість критичних дефіцитів. Як зазначається у дослідженні щодо скорочення часу виробничого циклу, подібний підхід до наскрізного моніторингу дозволяє виявити приховані неефективності ще на ранніх етапах планування, що суттєво зменшує ризики простоїв [4].

Важливо, що розроблена модель не лише розраховує матеріальні потреби, але й динамічно перерозподіляє ресурси між виробничими лініями у разі виникнення нестандартних ситуацій. Це відповідає завданню щодо оптимізації розміщення деталей, де основною метою є досягнення балансу між швидкістю обслуговування запитів та завантаженістю потужностей обладнання [3]. Таким чином, імітаційний підхід підтверджує, що розроблена система здатна до самостійної адаптації та ефективного керування транспортними потоками в автоматизованому середовищі, що є ключовим показником для сучасних виробництв [2].

3.5. Інформаційне забезпечення роботи автоматизованої системи

Інформаційне забезпечення автоматизованої системи керування потоками деталей призначене для накопичення, зберігання, обробки та передачі даних між окремими функціональними підсистемами. Воно забезпечує підтримку процесів контролю якості деталей, визначення пріоритетів розподілу, обліку складських запасів та контролю виконання виробничого плану.

Основою інформаційного забезпечення є база даних, яка містить відомості про номенклатуру деталей, складальні дільниці, виробничі плани, поточний стан складу та результати роботи системи. Формування та обробка цих даних здійснюється програмними модулями керуючої програми, опис яких наведено у додатку Б.

Для демонстрації структури інформаційного забезпечення розглянемо модель **FPV-дрона Mark4-Light**. Для даної моделі в системі зберігається перелік комплектуючих виробу із зазначенням шифру, найменування та необхідної кількості деталей для однієї збірки. Приклад збірки (рис. 3.15) та специфікації наведено у таблиці 3.5.

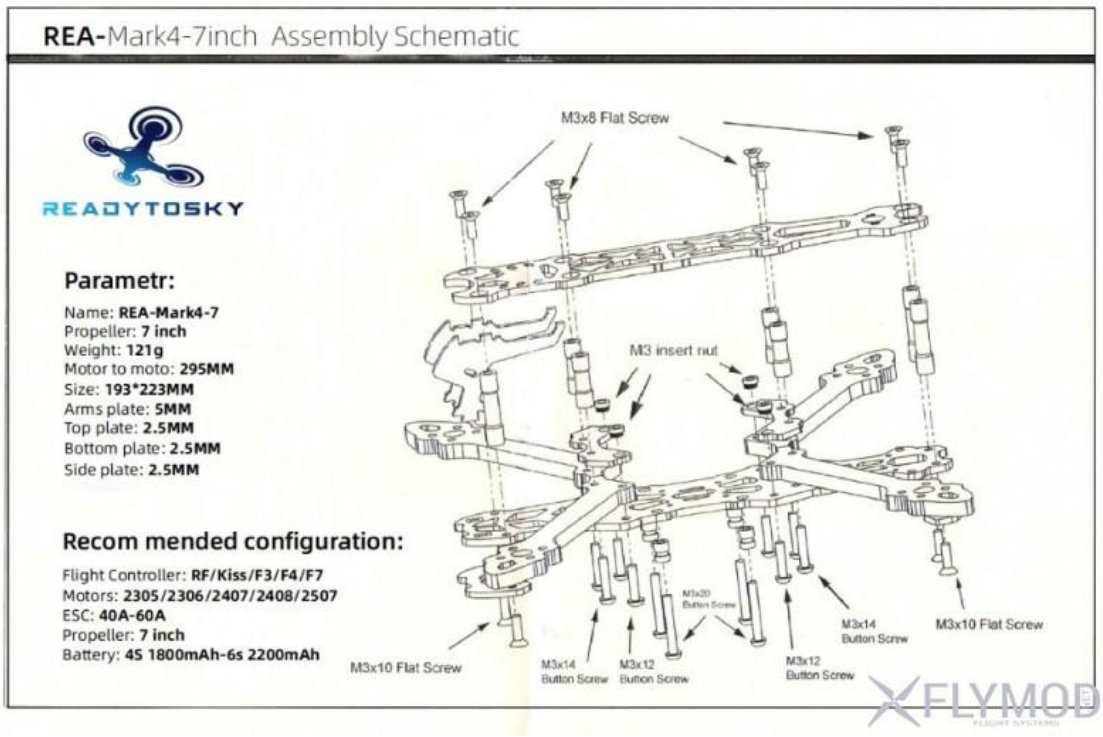


Рис. 3.14. Mark4 Light [10]

Таблиця 3.5. Специфікація моделі Mark4-Light

Mark4-Light (код: L)		
Шифр	Найменування деталі	Кількість штук
L-RM-001	Рама Mark4-Light (полегшена)	1
L-EL-001	Польотний стек (FC+ESC) – економ-клас	1
L-MT-001	Двигуни – базовий клас	4
L-PR-001	Пропелери – стандартні	4
L-VC-001	Відеосистема – аналог (базова)	1
U-HD-001	Гвинт M3×6 мм	4

U-HD-002	Гвинт M3×14 мм	2
U-RC-001	Радіоприймач ELRS 2.4 GHz	1
U-OT-001	Ремінець акумулятора Velcro	1
U-EL-001	Конденсатор живлення Low ESR	1
U-EL-002	Кабель живлення з роз'ємом XT60	1
U-HD-001	Гвинт M3×6 мм	4
U-HD-002	Гвинт M3×14 мм	2
U-RC-001	Радіоприймач ELRS 2.4 GHz	1
U-OT-001	Ремінець акумулятора Velcro	1
U-EL-001	Конденсатор живлення Low ESR	1
U-EL-002	Кабель живлення з роз'ємом XT60	1

У структурі бази даних кожна деталь характеризується унікальним шифром, належністю до певної категорії, кількістю на складі та переліком моделей, для яких допускається її використання. Такий підхід дозволяє реалізувати механізм перенаправлення деталей після проходження контролю якості. Якщо параметри деталі не відповідають вимогам моделі вищого класу, але залишаються допустимими для моделі нижчого рівня, система може автоматично змінити маршрут її подальшого використання.

Інформаційні потоки в автоматизованій системі організовані таким чином, щоб забезпечити постійний зворотний зв'язок між складом, підсистемою контролю якості та складальними дільницями. Після надходження деталі зі складу інформація про її переміщення фіксується в журналі подій. Далі система виконує контроль якості та визначає один із можливих маршрутів: передача на складальну дільницю, направлення на доопрацювання, перенаправлення на іншу модель або списання на переробку. Усі результати також реєструються в базі даних.

Для визначення пріоритету передачі деталей використовується спеціалізований модуль розрахунку пріоритетів **PriorityEngine** (додаток Б). Його робота базується на аналізі поточної потреби складальної дільниці, стратегічного пріоритету моделі та характеристик самої деталі. У результаті система автоматично обирає дільницю, яка найбільше потребує відповідної комплектуючої.

Окрему роль в інформаційному забезпеченні відіграє журнал виробничих подій. У ньому накопичуються записи про приймання деталей, результати контролю якості, випадки перенаправлення, виникнення дефіциту комплектуючих та завершення складання виробів. Така інформація використовується для аналізу роботи системи та оцінювання ефективності прийнятих алгоритмічних рішень.

Для формування виробничої звітності та статистичної обробки результатів у програмному забезпеченні використовуються засоби бібліотеки **Pandas**, призначеної для роботи з табличними даними [8]. Генерація випадкових виробничих подій та статистичне моделювання виконуються із застосуванням можливостей бібліотеки **NumPy** [9]. Організація черг та вибір пріоритетних елементів реалізовані відповідно до принципів роботи пріоритетних структур даних, описаних у документації Python [7].

Таким чином, інформаційне забезпечення розробленої автоматизованої системи формує єдиний інформаційний простір для всіх етапів виробничого процесу. Це забезпечує оперативний контроль руху деталей, підтримує механізми пріоритетного розподілу ресурсів та створює інформаційну основу для прийняття керуючих рішень у режимі реального часу.

3.6. Інструкція користувача при роботі з системою автоматизованого контролю

Програма призначена для контролю транспортування та обліку деталей під час складання FPV-дронів Mark4 трьох моделей: Mark4-Pro, Mark4-Standard та Mark4-Light. Вона дозволяє задати план виробництва, автоматично розрахувати необхідну кількість деталей, відстежувати їх розподіл між складом і дільницями, а також переглядати журнал подій під час виконання симуляції.

Перед початком роботи користувач **вводить кількість збірок** для кожної моделі у відповідні поля: Mark4-Pro, Mark4-Standard, Mark4-Light. Після введення значень програма автоматично формує перелік потрібних деталей із зазначенням їх шифру, назви та кількості. Це дає змогу одразу перевірити, які комплектуючі необхідні для виконання заданого плану.

Для запуску роботи системи потрібно натиснути кнопку **Запустити**. Після цього програма починає моделювати подачу деталей зі складу, їх розподіл за дільницями та процес накопичення комплектів для складання. У правій частині вікна відображається **журнал подій**, у якому фіксуються прийняті деталі, перенаправлення, випадки браку, готові збірки та інші важливі стани системи.

Кнопка **Стоп** використовується для зупинки поточної симуляції. При цьому вже отримані результати залишаються на екрані, тому користувач може проаналізувати стан складу, дільниць, журналу та статистики на момент зупинки.

Кнопка **Скинути** повертає систему до початкового стану відповідно до введеного плану. Вона очищає результати попереднього запуску, оновлює склад, буфери дільниць і журнал, після чого можна починати нову симуляцію.

Кнопка **Очистити журнал** видаляє записи журналу подій. Вона не змінює кількість деталей на складі, стан дільниць або кількість уже зібраних виробів. Ця функція потрібна, якщо користувач хоче прибрати зайві повідомлення з журналу без повного перезапуску системи.

Кнопка **Оновити** примусово оновлює інформацію у всіх таблицях інтерфейсу. Вона використовується для перегляду актуального стану складу, дільниць, готових збірок, статистики та журналу.

У нижній частині правого вікна розміщені вкладки **Склад, Дільниці, Збірки** та **Статистика**. Вкладка **Склад** показує залишки деталей. Вкладка **Дільниці** відображає деталі, які вже передані на відповідні виробничі дільниці. Вкладка **Збірки** містить інформацію про заплановану та фактично зібрану кількість виробів. Вкладка **Статистика** дозволяє оцінити середній час складання по кожній дільниці.

Після завершення симуляції програма формує **підсумковий звіт**. У ньому вказується кількість зібраних виробів кожної моделі, кількість деталей, що були відправлені на доопрацювання або переробку, загальний час виконання та середній час складання на дільницях. Якщо план не виконано через нестачу деталей, система **повідомляє**, для якої моделі виникла проблема та яких деталей не вистачає.

ВИСНОВКИ ДО РОЗДІЛУ

У цьому розділі було розроблено програмне забезпечення автоматизованої системи керування потоками деталей для складальних дільниць виробництва FPV-дронів. На основі поставленої задачі сформовано структуру програмного комплексу, реалізовано алгоритми контролю якості деталей, розподілу матеріальних потоків та підтримки виробничого плану.

У процесі розробки було створено модулі керування базою даних, моделювання виробничого процесу, визначення пріоритетів складальних дільниць та графічний інтерфейс користувача. Реалізований алгоритм дозволяє автоматично аналізувати стан складських запасів, виконувати контроль якості деталей, приймати рішення щодо їх подальшого маршруту та розподіляти комплектуючі між дільницями відповідно до поточних виробничих потреб.

Особливістю розробленої системи є використання багаторівневого механізму пріоритетів, який враховує дефіцит деталей на складальних дільницях, стратегічну важливість моделей та характеристики комплектуючих. Це дозволяє забезпечити більш ефективне використання наявних ресурсів і знизити ризик простою складальних дільниць через нестачу необхідних деталей.

Також було розроблено інформаційне забезпечення системи, що включає структуру бази даних, довідники деталей для моделей Mark4-Pro, Mark4-Standard та Mark4-Light, журнали виробничих подій і механізми обліку переміщення комплектуючих між функціональними підсистемами. Створене інформаційне середовище забезпечує цілісність даних та підтримує прийняття керуючих рішень у режимі реального часу.

Таким чином, у результаті виконання третього розділу було створено програмну модель автоматизованої системи транспортування та розподілу деталей, яка реалізує принципи динамічного керування матеріальними потоками та може бути використана для подальшого дослідження ефективності запропонованих алгоритмів у четвертому розділі дипломної роботи.

ВИСНОВКИ

У дипломній роботі вирішено задачу Автоматизація керування потоками деталей у багаторівневій транспортній системі з використанням пріоритетних черг на прикладі складання FPV-дронів. Актуальність роботи обумовлена необхідністю підвищення ефективності виробничих процесів шляхом оптимізації розподілу комплектуючих та зменшення простоїв складальних дільниць, які виникають через нерівномірне забезпечення деталями.

У процесі виконання роботи було проведено аналіз сучасних автоматизованих транспортних систем, методів керування матеріальними потоками та підходів до оптимізації виробничої логістики. На його основі встановлено доцільність використання елементів теорії масового обслуговування та механізмів пріоритетних черг для організації розподілу деталей між складальними дільницями.

Розроблено структуру автоматизованої системи керування, яка забезпечує контроль руху комплектуючих від складу до складальних дільниць, підтримує багаторівневий механізм визначення пріоритетів та враховує результати контролю якості деталей. Запропонований алгоритм дозволяє автоматично приймати

рішення щодо маршруту переміщення комплектуючих залежно від поточних потреб виробництва та рівня забезпеченості дільниць.

У межах роботи створено програмне забезпечення, імітаційна модель, яка реалізує функції обліку деталей, формування виробничого плану, визначення пріоритетів розподілу, ведення журналу виробничих подій та моделювання процесу складання виробів. Також розроблено інформаційне забезпечення системи, що включає структуру бази даних, довідники комплектуючих та механізми накопичення статистичної інформації.

Проведене моделювання підтвердило працездатність запропонованих алгоритмів та можливість їх використання для підтримки процесів керування матеріальними потоками. Використання динамічної системи пріоритетів дозволяє більш ефективно розподіляти ресурси між складальними дільницями та своєчасно реагувати на зміни виробничої ситуації.

Таким чином, поставлену мету дипломної роботи досягнуто. Розроблена автоматизована система може бути використана як основа для подальшого розвитку засобів інтелектуального керування виробничими процесами та дослідження методів оптимізації логістики в автоматизованих виробничих системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Поткін О. О., Тубальцева Н. П., Фатєєв М. В. Логістика та управління ланцюгами постачань у виробничих системах : методичні вказівки для самостійної роботи слухачів. Миколаїв : НУК, 2023. 132 с.
2. Гацько М. В., Шевченко В. В. Оптимізація транспортування деталей в умовах автоматизованого середньосерійного виробництва. Високоефективні технологічні процеси в приладобудуванні : Вісник НТУУ «КПІ». 2024.
3. Дмитренко С. А., Барандич К. С. Багатокритеріальна оптимізація розміщення деталей в умовах автоматизованого складу. Високоефективні технологічні процеси в приладобудуванні. Серія Приладобудування. К. : НТУУ «КПІ», 2025. Вип. 69(1).
4. Reducing Lead Times in High-Volume Manufacturing Process. Strategies and Case Studies. Olanab Consults, 2024.
5. Яганов П. О. Дослідження систем масового обслуговування : тексти лекцій. К. : НТУУ «КПІ», 2006. 40 с.
6. Buzacott J. A., Yao D. D. On queueing network models of flexible manufacturing systems. Queueing Systems. 1986. Vol. 1. P. 5–27.
7. Python 3.14.5 documentation. heapq — Heap queue algorithm. URL: https://docs.python.org/3/library/heapq.html (дата звернення: 08.06.2026).
8. pandas 3.0.3 documentation. User Guide. URL: https://pandas.pydata.org/docs/user_guide/index.html (дата звернення: 08.06.2026).
9. NumPy User Guide. Release 2.4.0. NumPy Documentation, 2024. URL: https://numpy.org/doc/ (дата звернення: 08.06.2026).
10. RCDrone. Mark4 5inch FPV Frame Kit. URL: https://rcdrone.top/uk/products/mark4-5inch-fpv-frame-kit-uk (дата звернення: 09.06.2026).
11. Шевченко І. А. Автоматизована система керування чергою видачі деталей на виробничій лінії : кваліфікаційна робота бакалавра / КПІ ім. Ігоря Сікорського. Київ, 2025. URL: https://ela.kpi.ua/items/7a2c7d2a-1d67-47d6-9a2c-6591a4557645 (дата звернення: 09.06.2026).
12. Kleinrock L. Queueing Systems. Volume 1: Theory. New York : Wiley-Interscience, 1975. 417 p.
13. tkinter — Graphical User Interfaces. Python 3.14.5 documentation. URL: https://docs.python.org/3/library/tkinter.html (дата звернення: 08.06.2026).
14. sqlite3 — DB-API 2.0 interface for SQLite databases. Python 3.14.5 documentation. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 08.06.2026).

15. Кубрак А. І., Жученко А. І., Кваско М. З. Комп'ютерне моделювання та ідентифікація автоматичних систем : навч. посіб. для студ. вищих навч. закл. К. : Політехніка, 2004. 424 с. URL: https://opac.kpi.ua/F/?func=direct&doc_number=000160477&local_base=KPI01 (дата звернення: 09.06.2026).
16. ДСТУ EN ISO 1302:2018. Технічні вимоги до геометричних характеристик продукції (GPS). [Чинний від 2019-01-01]. Вид. офіц. Київ : ДП «УкрНДНЦ», 2018.
17. Румбешта В. О. Основи технології складання приладів : підручник. К. : ІСДО, 2013. 303 с.
18. Стельмах Н. В. Технології складання в автоматизованому виробництві. Комп’ютерний практикум / КПІ ім. Ігоря Сікорського. Київ, 2023. 77 с.
19. Комп'ютерне моделювання: процеси і системи : підручник для студ. спеціальності «Автоматизація та комп’ютерно-інтегровані технології» / уклад.: І. В. Кравченко, І. В. Микитенко, Г. С. Тимчик ; КПІ ім. Ігоря Сікорського. Київ, 2022. 215 с. URL: https://ela.kpi.ua/handle/123456789/48860 (дата звернення: 09.06.2026).
20. Автоматизація виробничих процесів : навчальний посібник / Я. І. Проць, В. Б. Савків, О. К. Шкодзінський, О. Л. Ляшук ; за ред. І. Я. Проця. Тернопіль : ТНТУ, 2011. 344 с.
21. Пономарьов В. Д. Переваги автоматизації виробничих процесів. Інформаційні технології: наука, техніка, технологія, освіта, здоров’я : тези доповідей 32-ї міжнародної науково-практичної конференції MicroCAD–2024 (Харків, 2024 р.). Харків : НТУ «ХПІ», 2024. С. 862.
22. Groover M. P. Automation, Production Systems, and Computer-Integrated Manufacturing. 5th ed. Pearson, 2018. 912 p.
23. Невлюдов І. Ш. Виробничі процеси та обладнання об’єктів автоматизації: підручник. / І. Ш. Невлюдов : підручник для студентів вищих навчальних закладів. – Кривий Ріг : Криворізький коледж НАУ, 2017. – 444 с.

Додаток А

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології»

на тему: «Автоматизована система керування транспортними потоками деталей»

Виконала студентка групи ПБ-21 Хохич Аліна

Актуальність дослідження. Сучасне виробництво потребує ефективних засобів керування матеріальними потоками, оскільки швидкість і точність доставки деталей безпосередньо впливають на продуктивність складальних дільниць. Особливої актуальності це набуває в умовах одночасного виготовлення декількох модифікацій виробів, які можуть використовувати як спільні, так і спеціалізовані комплектуючі.

Традиційні системи транспортування деталей часто використовують фіксовані правила розподілу ресурсів, що не враховують поточний стан виробництва та рівень забезпеченості складальних дільниць. Це може призводити до виникнення дефіциту комплектуючих, простоїв обладнання та зниження ефективності виробничого процесу. Тому актуальним є розроблення автоматизованої системи керування потоками деталей із використанням механізмів пріоритетного розподілу ресурсів.

Метою дипломної роботи є розроблення автоматизованої системи керування потоками деталей між складальними дільницями, яка забезпечує пріоритетний розподіл комплектуючих відповідно до поточних потреб виробництва та дослідження ефективності її роботи шляхом програмного моделювання.

Для досягнення мети дослідження необхідно виконати такі теоретичні й практичні завдання:

– провести аналіз існуючих методів керування матеріальними потоками в автоматизованих виробничих системах;

– дослідити можливості застосування теорії масового обслуговування та пріоритетних черг для розподілу деталей;

– розробити структуру автоматизованої системи керування потоками деталей;

– створити алгоритм визначення пріоритетів складальних дільниць та розподілу комплектуючих;

– розробити інформаційне забезпечення та структуру бази даних системи;

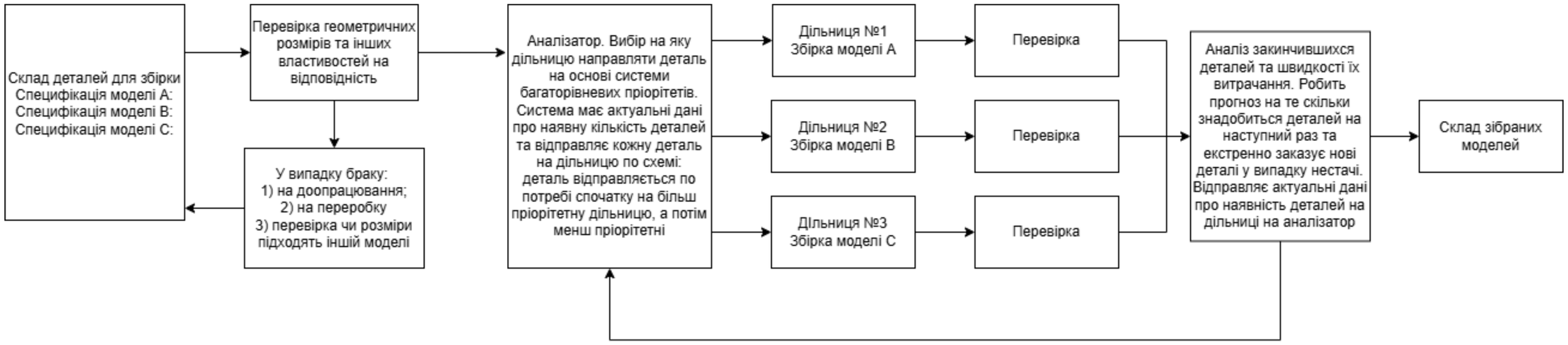
– реалізувати програмне забезпечення для моделювання виробничого процесу;

– виконати дослідження роботи системи в різних виробничих умовах та проаналізувати отримані результати.

Аналіз автоматизованих систем керування транспортними потоками деталей

Технічний засіб	Головні переваги (Плюси)	Основні недоліки (Мінуси)	Найкраща сфера застосування	Посилання на джерело
Стаціонарні конвеєри (рольганги, стрічки)	Безперервна подача, дуже висока продуктивність на стабільних потоках.	Немає маневреності, займають багато місця, важко перепла нувати цех.	Масове та великосерійне виробництво однотипних деталей.	[1, 2]
Автоматизовані візки (AGV/AMR)	Висока манереність (особливо на колесах меканум), легка зміна маршрутів.	Обмежена вантажопідйомність, залежність від заряду акумулятора.	Гнучке середньосерійне виробництво приладобудування.	[2]
Класичні склади з ручним пошуком	Низька вартість обладнання, простота організації процесу.	Повільний пошук деталей, часті помилки через людський фактор.	Невеликі майстерні або дрибносерійне виробница	[1, 3]
Автоматизовані висотні склади (AS/RS)	Максимальне використання висоти цеху, швидкий автоматичний пошук без помилок.	Дуже висока вартість обладнання та складне обслуговування.	Великі логістичні центри, сучасні автоматизовані заводи.	[3]

Структурно-функціональна схема автоматизованої системи



Математична модель багаторівневої системи керування з використанням пріоритетних черг

1. Інтенсивність надходження деталей до системи:

$$\lambda = \frac{1}{t_{\text{надход}}}$$

Де:

λ – інтенсивність надходження деталей (кількість заявок за одиницю часу);

$t_{\text{надход}}$ – середній час між надходженнями двох послідовних деталей.

2. Інтенсивність обслуговування деталей:

$$\mu = \frac{1}{t_{\text{обс}}}$$

Де:

μ – інтенсивність обслуговування (кількість деталей, що можуть бути оброблені системою за одиницю часу);

$t_{\text{обс}}$ – середній час виконання однієї технологічної операції.

3. Коефіцієнт завантаження системи:

$$\rho = \frac{\lambda}{n\mu} < 1$$

Де:

ρ – коефіцієнт завантаження системи (показує ступінь використання потужностей);

λ – інтенсивність надходження деталей;

n — кількість каналів обслуговування (складальних дільниць);

μ — інтенсивність обслуговування.

4. Ймовірність вільного стану системи (всі канали вільні):

$$P_0 = \left[\sum_{h=0}^{n} \frac{(n\rho)^k}{k!} + \frac{(n\rho)^{n+1}}{n!\,(1-\rho)}\right]$$

Де:

P_0 – ймовірність того, що всі канали обслуговування вільні;

n – кількість каналів обслуговування;

ρ – коефіцієнт завантаження системи;

k – індекс сумування (кількість зайнятих каналів).

5. Ймовірність очікування (всі канали зайняті):

$$P_{\text{очік}} = \frac{(n\rho)^n}{n!\,(1-\rho)} P_0$$

Де:

$P_{\text{очік}}$ – ймовірність того, що деталь потрапить у чергу (всі канали зайняті);

n – кількість каналів обслуговування;

ρ – коефіцієнт завантаження системи;

P_0 – ймовірність вільного стану системи (з формули 4).

6. Інтегральний пріоритет деталі:

$$W(i,j) = \alpha P_{\text{ПрПотр}}(j) + \beta P_{\text{ПрДет}}(i) + \gamma P_{\text{ПрДільн}}(j)$$

Де:

$W(i,j)$ – інтегральний (узагальнений) пріоритет i-тої деталі на j-й дільниці;

$P_{\text{ПрПотр}}(j)$ – показник пріоритету за потребою на дільниці j;

$P_{\text{ПрДег}}(i)$ – показник пріоритету за легкістю обробки деталі i;

$P_{\text{ПрДільн}}(j)$ – показник пріоритету за лінійною безпекою/стабільністю дільниці j;

α, β, γ – вагові коефіцієнти, що визначають важливість кожного критерію.

7. Середній час очікування для деталі k-го пріоритетного класу:

$$t_{\text{очік}}^{k} = \frac{\sum_{s=1}^{R} \lambda t_{\text{обс},s}^{2}}{2\left(1 - \sum_{s=1}^{k-1} \rho_s\right) - \left(1 - \sum_{s=1}^{k} \rho_s\right)}$$

Де:

$t_{\text{очік}}^{k}$ – середній час очікування деталі, що належить до k-го пріоритетного класу;

λ – інтенсивність потоку;

$t_{\text{обс},s}$ – час обслуговування для класу s;

ρ_s – коефіцієнт завантаження для класу s;

R – загальна кількість пріоритетних класів.

8. Повна інтенсивність потоку через j-й елемент:

$$\Lambda_j = \lambda_j + \sum_{k=1}^{K} \Lambda_k p_{kj}$$

Де:

Λ_j – сумарна інтенсивність потоку через j-й елемент системи;

λ_j – зовнішній потік заявок, що надходить безпосередньо на j-й елемент;

λ_k інтенсивність потоку, що приходить із k-го елемента;

p_{kj} — ймовірність переходу деталі від k-го до j-го елемента.

9. Цільова функція керування транспортними потоками:

$$F = \min\left[\sum_{i=1}^{N} \sum_{j=1}^{M} t_{\text{трансп}}(i,j) \cdot x(i,j) + \sum_{j=1}^{M} \Delta t_{\text{простой}}(j) \cdot W_{\text{ПоточПр}}(j)\right]$$

Де:

F – цільова функція (яку потрібно мінімізувати);

N – загальна кількість деталей;

M – кількість складальних дільниць;

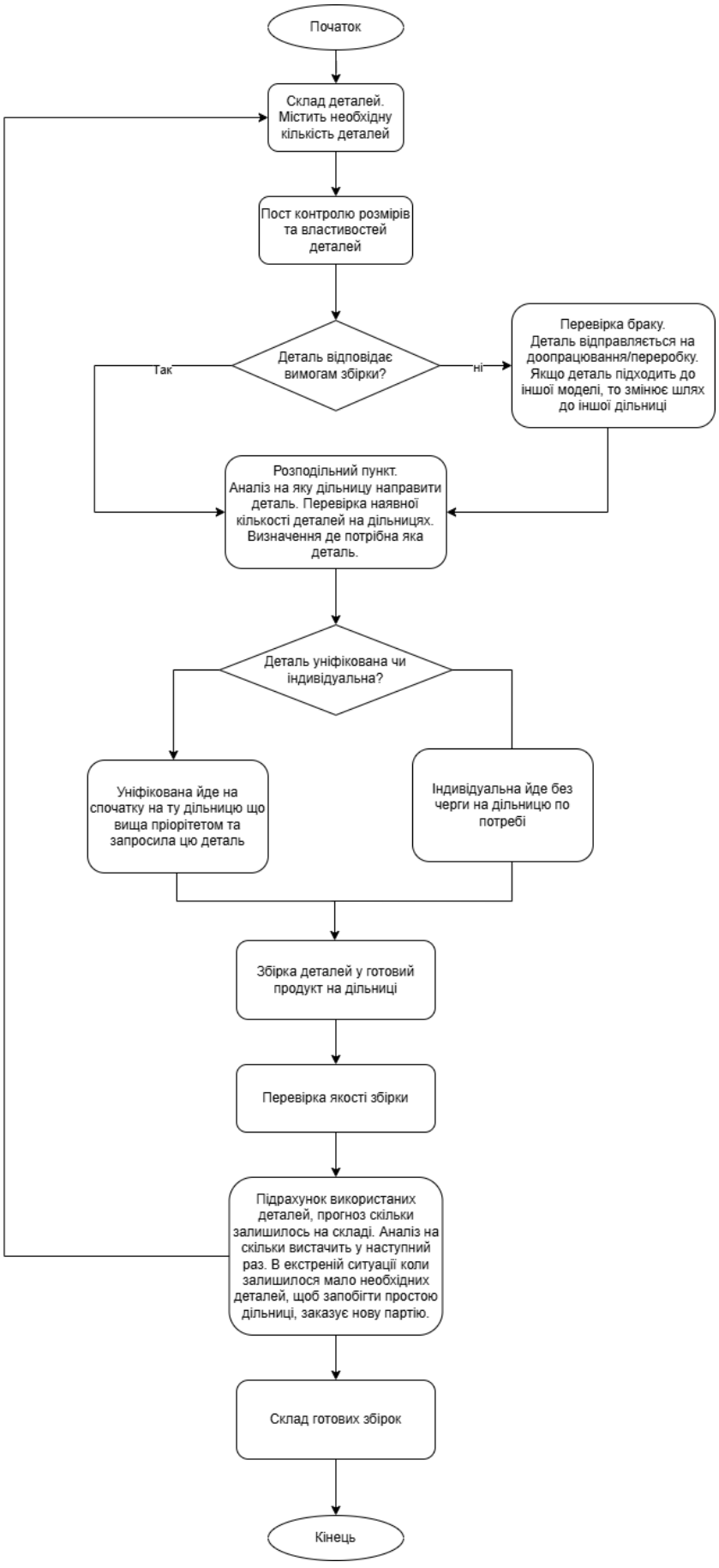
$t_{\text{трансп}}(i,j)$ – час транспортування i-тої деталі до j-ї дільниці;

$x(i,j)$ булева змінна керування (1, якщо деталь направлено на дільницю, 0 – в іншому випадку);

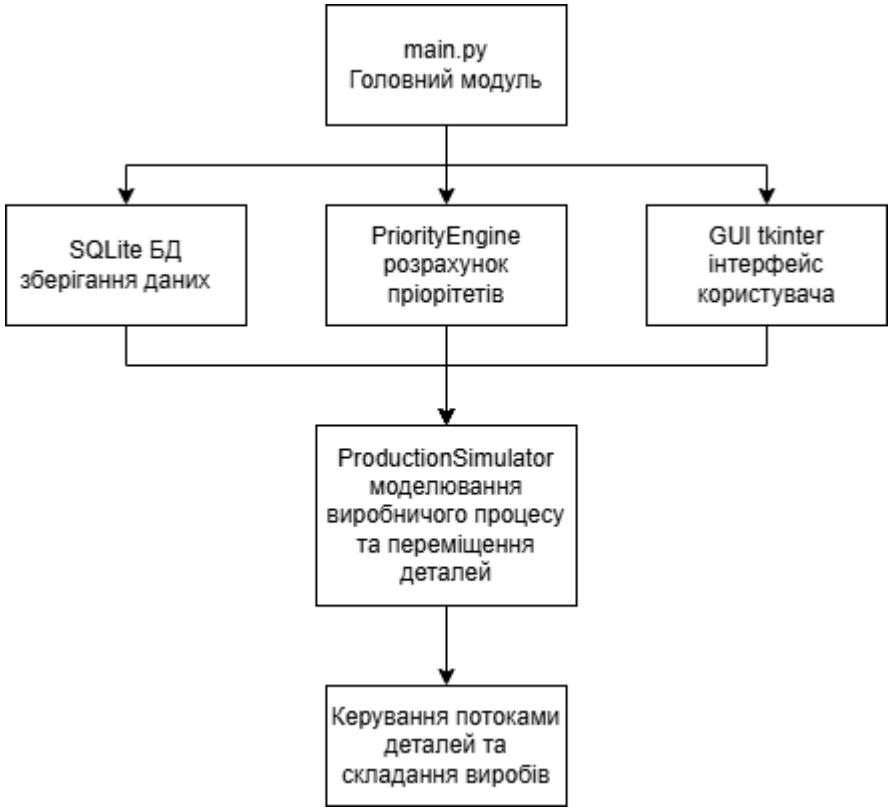
$\Delta t_{\text{простой}}(j)$ – час вимушеного простою j-ї дільниці;

$W_{\text{ПоточПр}}(j)$ – ваговий коефіцієнт або інтенсивність потоку, що впливає на простій j-ї дільниці.

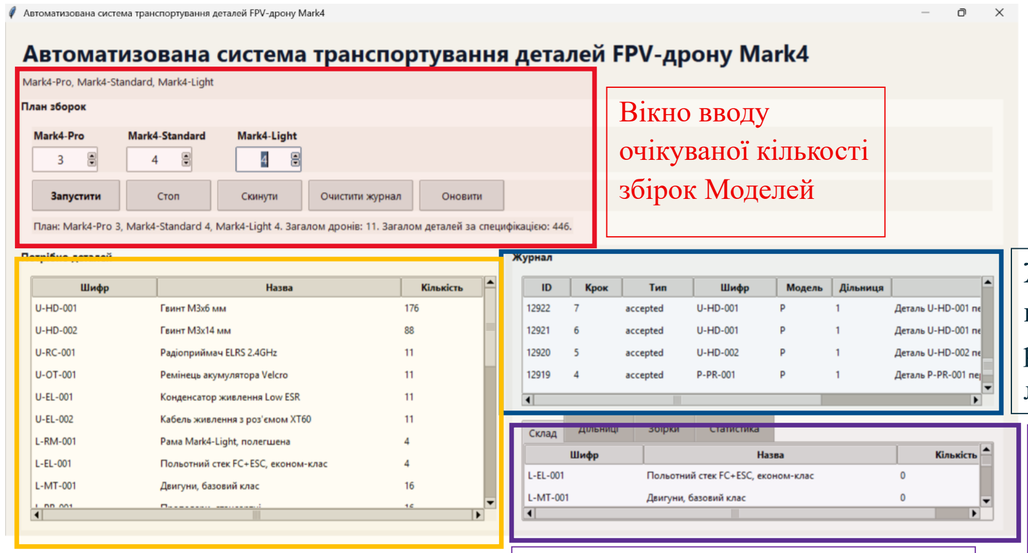
Алгоритм керування з врахуванням пріоритетних черг



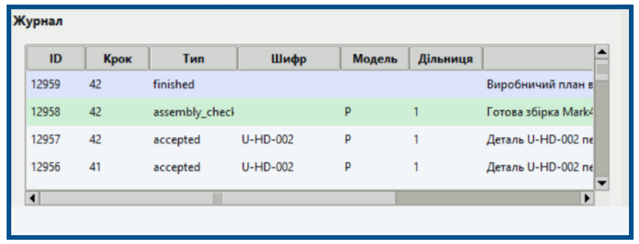
Структура програмного забезпечення автоматизованої системи



Приклад функціонування автоматизованої системи керування



Вікно вводу очікуваної кількості збірок Моделей



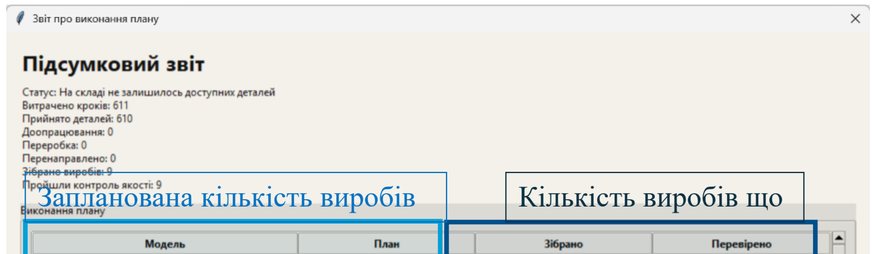
Приклад завершення збірки та завершення роботи

Автоматичний підрахунок потрібної кількості деталей згідно плану

Журнал де відображається робота транспортної лінії

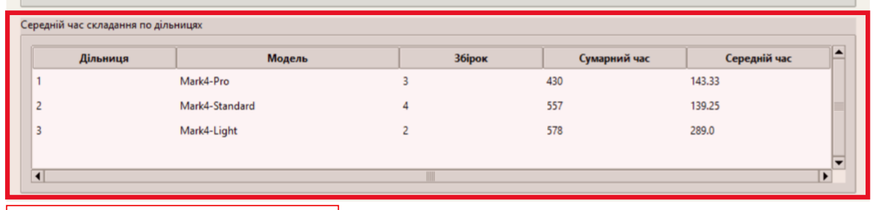
Склад деталей де видно наявну кількість деталей на момент фото

Набір вкладок де можна динамічно відслідковувати кількість деталей на різних точках системи

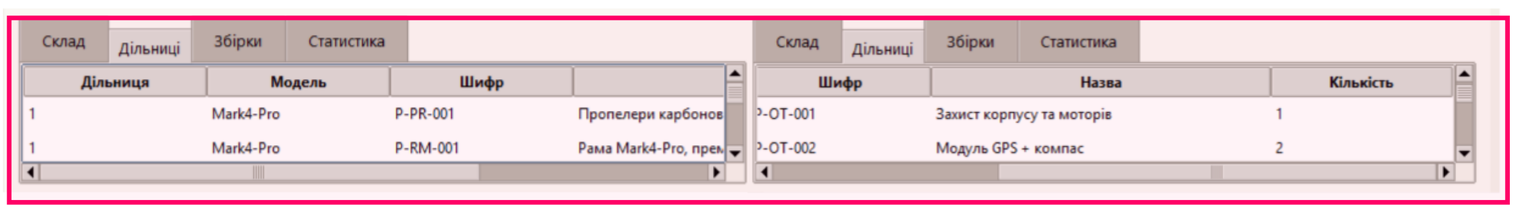


Запланована кількість виробів

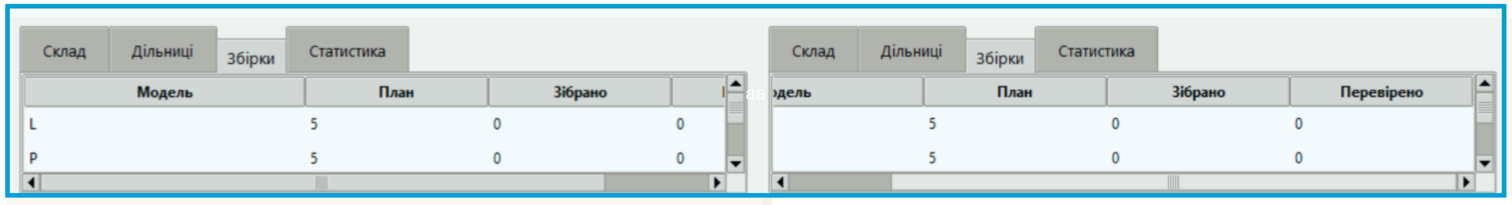
Кількість виробів що



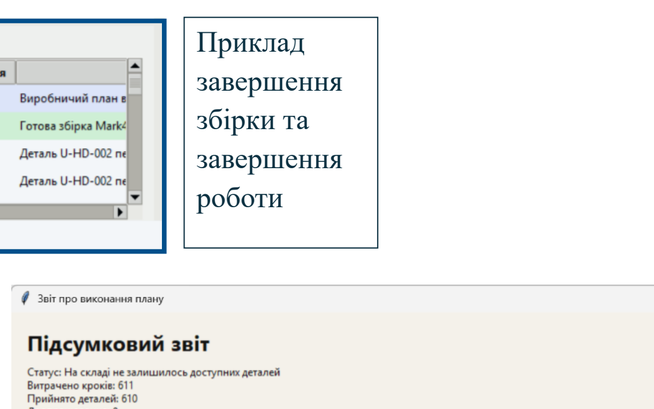
Підсумкова статистика



Вкладка «Дільниці», де відображається кількість деталей на кожній з дільниць

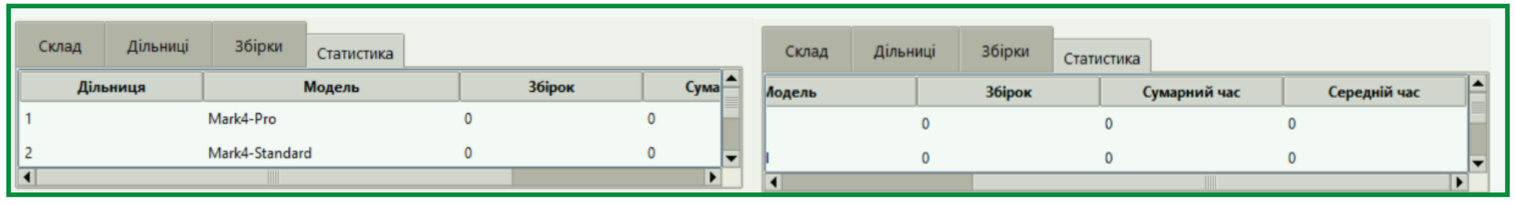


Вкладка «Збірки», де відображається кількість зібраних виробів та їх перевірка



Повідомлення про нестачу деталей та порушення виробничого плану.

Підрахунок бракованих деталей



Вкладка «Статистика», де відображаються підсумки роботи, тобто кількість готових виробів та час їх перебування на дільниці

Інформаційне забезпечення автоматизованої системи

Інформаційне забезпечення автоматизованої системи керування потоками деталей призначене для накопичення, зберігання, обробки та передачі даних між окремими функціональними підсистемами. Воно забезпечує підтримку процесів контролю якості деталей, визначення пріоритетів розподілу, обліку складських запасів та контролю виконання виробничого плану.

Основою інформаційного забезпечення є база даних, яка містить відомості про номенклатуру деталей, складальні дільниці, виробничі плани, поточний стан складу та результати роботи системи. Формування та обробка цих даних здійснюється програмними модулями керуючої програми (додаток Б).

Для демонстрації структури інформаційного забезпечення розглянемо модель **FPV-дрона Mark4-Light**. Для даної моделі в системі зберігається перелік комплектуючих виробу із зазначенням шифру, найменування та необхідної кількості деталей для однієї збірки. Приклад збірки (на рис. 1) та специфікації наведено у таблиці 1.

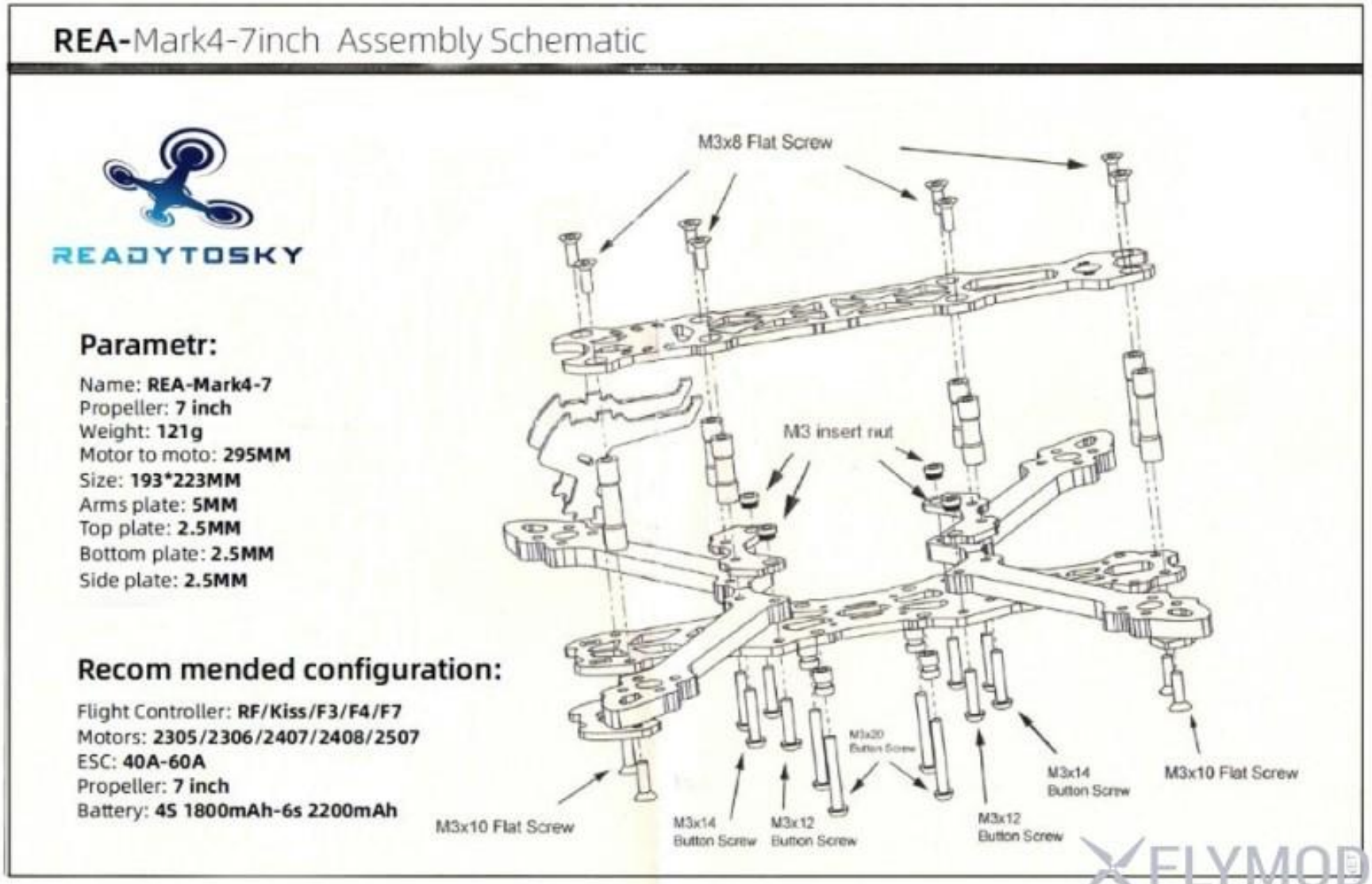


Рис. 1 Mark4 Light [10]

Таблиця 1. Специфікація моделі Mark4-Light

Модель Mark4-Light								
Шифр	Найменування деталі	Кількість штук	Шифр	Найменування деталі	Кількість штук	Шифр	Найменування деталі	Кількість штук
L-RM-001	Рама Mark4-Light (полегшена)	1	U-HD-002	Гвинт M3×14 мм	2	U-HD-002	Гвинт M3×14 мм	2
L-EL-001	Польотний стек (FC+ESC) – економ-клас	1	U-RC-001	Радіоприймач ELRS 2.4 GHz	1	U-RC-001	Радіоприймач ELRS 2.4 GHz	1
L-MT-001	Двигуни – базовий клас	4	U-OT-001	Ремінець акумулятора Velcro	1	U-OT-001	Ремінець акумулятора Velcro	1
L-PR-001	Пропелери – стандартні	4	U-EL-001	Конденсатор живлення Low ESR	1	U-EL-001	Конденсатор живлення Low ESR	1
L-VC-001	Відеосистема – аналог (базова)	1	U-EL-002	Кабель живлення з роз'ємом XT60	1	U-EL-002	Кабель живлення з роз'ємом XT60	1
U-HD-001	Гвинт M3×6 мм	4	U-HD-001	Гвинт M3×6 мм	4			

У структурі бази даних кожна деталь характеризується унікальним шифром, належністю до певної категорії, кількістю на складі та переліком моделей, для яких допускається її використання. Такий підхід дозволяє реалізувати механізм перенаправлення деталей після проходження контролю якості. Якщо параметри деталі не відповідають вимогам моделі вищого класу, але залишаються допустимими для моделі нижчого рівня, система може автоматично змінити маршрут її подальшого використання.

Інформаційні потоки в автоматизованій системі організовані таким чином, щоб забезпечити постійний зворотний зв'язок між складом, підсистемою контролю якості та складальними дільницями. Після надходження деталі зі складу інформація про її переміщення фіксується в журналі подій. Далі система виконує контроль якості та визначає один із можливих маршрутів: передача на складальну дільницю, направлення на доопрацювання, перенаправлення на іншу модель або списання на переробку. Усі результати також реєструються в базі даних.

Для визначення пріоритету передачі деталей використовується спеціалізований модуль розрахунку пріоритетів **PriorityEngine** (додаток Б). Його робота базується на аналізі поточної потреби складальної дільниці, стратегічного пріоритету моделі та характеристик самої деталі. У результаті система автоматично обирає дільницю, яка найбільше потребує відповідної комплектуючої.

Окрему роль в інформаційному забезпеченні відіграє журнал виробничих подій. У ньому накопичуються записи про приймання деталей, результати контролю якості, випадки перенаправлення, виникнення дефіциту комплектуючих та завершення складання виробів. Така інформація використовується для аналізу роботи системи та оцінювання ефективності прийнятих алгоритмічних рішень.

Для формування виробничої звітності та статистичної обробки результатів у програмному забезпеченні використовуються засоби бібліотеки **Pandas**, призначеної для роботи з табличними даними [8]. Генерація випадкових виробничих подій та статистичне моделювання виконуються із застосуванням можливостей бібліотеки **NumPy** [9]. Організація

черг та вибір пріоритетних елементів реалізовані відповідно до принципів роботи пріоритетних структур даних, описаних у документації Python [7].

Таким чином, інформаційне забезпечення розробленої автоматизованої системи формує єдиний інформаційний простір для всіх етапів виробничого процесу. Це забезпечує оперативний контроль руху деталей, підтримує механізми пріоритетного розподілу ресурсів та створює інформаційну основу для прийняття керуючих рішень у режимі реального часу.

Додаток Б

Програмний код

main.py

```
from fpv_control.database import Database
from fpv_control.gui import run_app
from pathlib import Path


def main() -> None:
    base_dir = Path(__file__).resolve().parent
    db = Database(str(base_dir / "data" / "fpv_parts.sqlite"))
    db.initialize()
    db.seed_if_empty(target_units=50)
    run_app(db)


if __name__ == "__main__":
    main()
```

run_simulation.py

```
from fpv_control.database import Database
from fpv_control.simulator import ProductionSimulator
from pathlib import Path


def main() -> None:
    base_dir = Path(__file__).resolve().parent
    db = Database(str(base_dir / "data" / "fpv_parts.sqlite"))
    db.initialize()
    db.seed_if_empty(target_units=50)
    db.clear_log()
    db.reset_for_plan({"P": 50, "S": 50, "L": 50})

    simulator = ProductionSimulator(db)
    summary = simulator.run(steps=520)

    print("Simulation finished")
    print(f"Accepted parts: {summary.get('accepted', 0)}")
    print(f"Rework: {summary.get('rework', 0)}")
```

```
    print(f"Recycle: {summary.get('recycle', 0)}")
    print(f"Redirected: {summary.get('redirected', 0)}")
    print(f"Emergency shortages: {summary.get('shortages', 0)}")
    print(f"Assembled drones: {summary.get('assembled', 0)}")
    print(f"Passed final quality check: {summary.get('passed_quality', 0)}")


if __name__ == "__main__":
    main()
```

simulator.py

```
from __future__ import annotations

from collections import Counter
import random

try:
    import numpy as np
except ImportError:
    np = None

from .catalog import MODEL_BY_CODE, PARTS
from .priority import PriorityEngine


class ProductionSimulator:
    def __init__(self, db, defect_rate: float = 0.0001, seed: int | None = 42) -> None:
        self.db = db
        self.defect_rate = defect_rate
        self.rng = np.random.default_rng(seed) if np is not None else
random.Random(seed)
        self.priority = PriorityEngine(db)
        self.summary = Counter()
        self.tick = 0
        self.finished = False
        self.stop_reason = ""

    def run(self, steps: int = 100) -> dict[str, int]:
        for _ in range(steps):
```

```
            if not self.step():
                break
        return dict(self.summary)

    def run_until_finished(self, max_ticks: int = 20000, refresh_callback=None) ->
dict[str, int]:
        while self.tick < max_ticks and self.step():
            if refresh_callback and self.tick % 3 == 0:
                refresh_callback()
        if refresh_callback:
            refresh_callback()
        return dict(self.summary)

    def step(self) -> bool:
        if self.db.plan_completed():
            self.finished = True
            self.stop_reason = "План виконано"
            return False

        self.tick += 1
        part_code = self._random_available_part()
        if part_code is None:
            self._try_assemble_models(self.tick)
            if self.db.plan_completed():
                self.finished = True
                self.stop_reason = "План виконано"
            else:
                self.finished = True
                self.stop_reason = "На складі не залишилось доступних деталей"
                self.db.log(self.tick, "stop", self.stop_reason)
                self._log_shortage_report(self.tick)
            return False

        self.process_part(self.tick, part_code)
        self._try_assemble_models(self.tick)

        if self.db.plan_completed():
            self.finished = True
            self.stop_reason = "План виконано"
```

```
            self.db.log(self.tick, "finished", "Виробничий план виконано повністю")
            return False
        return True

    def process_part(self, tick: int, part_code: str) -> None:
        if not self.db.move_from_warehouse(part_code):
            self.summary["shortages"] += 1
            self.db.log(tick, "shortage", f"Недостача деталі {part_code} на складі",
part_code)
            return

        defect_result = self._quality_control()
        if defect_result == "accepted":
            self._route_accepted_part(tick, part_code)
            return
        if defect_result == "rework":
            self.summary["rework"] += 1
            self.db.log(tick, "rework", f"Деталь {part_code} направлено на
доопрацювання", part_code)
            return
        if defect_result == "recycle":
            self.summary["recycle"] += 1
            self.db.log(tick, "recycle", f"Деталь {part_code} списано на переробку",
part_code)
            return

        self._redirect_lower_model(tick, part_code)

    def _route_accepted_part(self, tick: int, part_code: str) -> None:
        section, model_code, score = self.priority.choose_section(part_code)
        self.db.add_to_section(section, part_code)
        self.summary["accepted"] += 1
        self.db.log(
            tick,
            "accepted",
            f"Деталь {part_code} передано на дільницю {section}; пріоритет {score}",
            part_code,
            model_code,
            section,
```

```
        )

    def _redirect_lower_model(self, tick: int, part_code: str) -> None:
        part = self.db.fetch_one("SELECT acceptable_models FROM parts WHERE code
= ?", (part_code,))
        if part is None:
            return
        # Імітуємо ситуацію, коли характеристики деталі недостатні для старшої
        # моделі, але допустимі для моделі нижчого класу.
        acceptable = tuple(code for code in ("L", "S") if code in
part["acceptable_models"])
        if not acceptable:
            self.summary["recycle"] += 1
            self.db.log(tick, "recycle", f"Деталь {part_code} не має допустимого
альтернативного маршруту", part_code)
            return
        section, model_code, score = self.priority.choose_section(part_code,
forced_models=acceptable)
        self.db.add_to_section(section, part_code)
        self.summary["redirected"] += 1
        self.db.log(
            tick,
            "redirected",
            f"Деталь {part_code} після контролю перенаправлено на модель
{model_code}; пріоритет {score}",
            part_code,
            model_code,
            section,
        )

    def _quality_control(self) -> str:
        if self._random_float() > self.defect_rate:
            return "accepted"
        return self._weighted_choice(["rework", "recycle", "redirect"], [0.45, 0.35, 0.20])

    def _random_available_part(self) -> str | None:
        rows = self.db.fetch_all("SELECT part_code, quantity FROM warehouse
WHERE quantity > 0")
        if not rows:
```

```
            return None
        codes = [row["part_code"] for row in rows]
        quantities = [row["quantity"] for row in rows]
        if np is not None:
            probabilities = np.array(quantities, dtype=float)
            probabilities = probabilities / probabilities.sum()
            return str(self.rng.choice(codes, p=probabilities))
        return self._weighted_choice(codes, quantities)

    def _try_assemble_models(self, tick: int) -> None:
        for model in MODEL_BY_CODE.values():
            goods = self.db.fetch_one("SELECT * FROM finished_goods WHERE
model_code = ?", (model.code,))
            if goods is None or goods["assembled"] >= goods["planned"]:
                continue
            assembled_now = 0
            while goods["assembled"] + assembled_now < goods["planned"]:
                plan = self._assembly_plan(model.code, model.section)
                if plan is None:
                    break
                for part_code, new_quantity in plan.items():
                    self.db.update_buffer_quantity(model.section, part_code, new_quantity)
                passed = bool(self._random_float() > 0.015)
                self.db.mark_assembled(model.code, passed)
                duration = self.db.record_assembly_time(model.section, tick)
                assembled_now += 1
                self.summary["assembled"] += 1
                if passed:
                    self.summary["passed_quality"] += 1
                self.db.log(
                    tick,
                    "assembly_check",
                    (
                        f"Готова збірка {model.name} на дільниці {model.section}. "
                        f"Час від попередньої готової збірки: {duration} хв."
                    ),
                    target_model=model.code,
                    target_section=model.section,
                )
```

```
    def _assembly_plan(self, model_code: str, section: int) -> dict[str, int] | None:
        buffers = self.db.fetch_all(
            """
            SELECT b.part_code, b.quantity, p.category, p.quality_rank
            FROM section_buffer b
            JOIN parts p ON p.code = b.part_code
            WHERE b.section = ? AND b.quantity > 0
            ORDER BY p.quality_rank DESC, b.quantity DESC
            """,
            (section,),
        )
        available = {row["part_code"]: dict(row) for row in buffers}
        new_quantities = {code: row["quantity"] for code, row in available.items()}
        requirements = [p for p in PARTS if p.model_code in ("U", model_code)]

        for required in requirements:
            remaining = required.required_per_drone
            candidates = self._candidate_parts_for_requirement(required, available)
            for candidate in candidates:
                code = candidate["part_code"]
                take = min(new_quantities[code], remaining)
                new_quantities[code] -= take
                remaining -= take
                if remaining == 0:
                    break
            if remaining:
                return None
        return new_quantities

    def _candidate_parts_for_requirement(self, required, available: dict[str, dict]) ->
list[dict]:
        if required.model_code == "U":
            row = available.get(required.code)
            return [row] if row else []
        exact = available.get(required.code)
        candidates = [exact] if exact else []
        substitutes = [
            row for row in available.values()
```

```
            if row["part_code"] != required.code
            and row["category"] == required.category
            and row["quality_rank"] >= required.quality_rank
            and not row["part_code"].startswith("U-")
        ]
        return candidates + substitutes

    def _log_shortage_report(self, tick: int) -> None:
        shortages = self.db.shortage_report()
        for model_name, items in shortages.items():
            detail_text = "; ".join(f"{code} - {qty} шт." for code, _name, qty in items)
            self.db.log(
                tick,
                "shortage_order",
                f"Кількість виробів {model_name} менша запланованої, заказано
недостатні деталі: {detail_text}",
            )

    def _random_float(self) -> float:
        return float(self.rng.random())

    def _weighted_choice(self, items: list[str], weights: list[float] | list[int]) -> str:
        if np is not None:
            probabilities = np.array(weights, dtype=float)
            probabilities = probabilities / probabilities.sum()
            return str(self.rng.choice(items, p=probabilities))
        return self.rng.choices(items, weights=weights, k=1)[0]
```

priority.py

```
from __future__ import annotations

import json
from dataclasses import dataclass



@dataclass(frozen=True)
class PriorityWeights:
    need: float = 0.58
    detail: float = 0.22
```

```
    section: float = 0.20


class PriorityEngine:
    def __init__(self, db, buffer_target: int = 6, critical_min: int = 2,
                 weights: PriorityWeights | None = None) -> None:
        self.db = db
        self.buffer_target = buffer_target
        self.critical_min = critical_min
        self.weights = weights or PriorityWeights()

    def choose_section(self, part_code: str, forced_models: tuple[str, ...] | None = None)
-> tuple[int, str, float]:
        part = self.db.fetch_one("SELECT * FROM parts WHERE code = ?",
(part_code,))
        if part is None:
            raise ValueError(f"Unknown part: {part_code}")
        acceptable = tuple(json.loads(part["acceptable_models"]))
        if forced_models:
            acceptable = tuple(code for code in acceptable if code in forced_models)
        if not acceptable:
            raise ValueError(f"Part {part_code} has no available target model")

        candidates = self.db.fetch_all(
            """
            SELECT m.code, m.section, m.strategic_priority,
                   COALESCE(b.quantity, 0) AS buffered
            FROM models m
            LEFT JOIN section_buffer b
              ON b.model_code = m.code AND b.part_code = ?
            WHERE m.code IN ({})
            """.format(",".join("?" for _ in acceptable)),
            (part_code, *acceptable),
        )

        best = None
        for row in candidates:
            need_score = self._need_score(row["buffered"], part["required_per_drone"])
            detail_score = part["quality_rank"] / 3
```

```
            section_score = row["strategic_priority"] / 3
            score = (
                self.weights.need * need_score
                + self.weights.detail * detail_score
                + self.weights.section * section_score
            )
            item = (score, row["section"], row["code"])
            if best is None or item > best:
                best = item

        assert best is not None
        score, section, model_code = best
        return section, model_code, round(score, 4)

    def _need_score(self, buffered: int, required_per_drone: int) -> float:
        critical_min = max(self.critical_min, required_per_drone)
        buffer_target = max(self.buffer_target, required_per_drone * 2)
        if buffered <= critical_min:
            return 1.0
        shortage = max(buffer_target - buffered, 0)
        return shortage / max(buffer_target, 1)
```

gui.py

```
from __future__ import annotations

import tkinter as tk
from tkinter import messagebox, ttk

from .catalog import PARTS
from .simulator import ProductionSimulator


APP_TITLE = "Автоматизована система транспортування деталей FPV-дрону Mark4"
MODELS_TEXT = "Mark4-Pro, Mark4-Standard, Mark4-Light"


class PartsControlApp(tk.Tk):
    def __init__(self, db) -> None:
```

```
        super().__init__()
        self.db = db
        self.simulator: ProductionSimulator | None = None
        self.running = False
        self.after_id: str | None = None

        self.title(APP_TITLE)
        self.geometry("1280x900")
        self.minsize(1100, 780)
        self.configure(bg="#f4f1ea")

        self.plan_vars = {
            "P": tk.IntVar(value=0),
            "S": tk.IntVar(value=0),
            "L": tk.IntVar(value=0),
        }
        self.plan_summary_var = tk.StringVar(value="")
        self.status_var = tk.StringVar(value="Система готова до введення плану")
        self.tables: dict[str, ttk.Treeview] = {}
        self.locked_controls: list[tk.Widget] = []

        self._configure_style()
        self._build_layout()
        self._bind_plan_updates()
        self._update_plan_preview()
        self.refresh()

    def _configure_style(self) -> None:
        style = ttk.Style(self)
        style.theme_use("clam")
        style.configure("TFrame", background="#f4f1ea")
        style.configure("TLabel", background="#f4f1ea", foreground="#111111")
        style.configure("Title.TLabel", font=("Segoe UI", 22, "bold"),
background="#f4f1ea", foreground="#0f172a")
        style.configure("Subtitle.TLabel", font=("Segoe UI", 10), background="#f4f1ea",
foreground="#111111")
        style.configure("Panel.TLabelframe", background="#faf8f3", relief="flat")
        style.configure(
            "Panel.TLabelframe.Label",
```

```
            background="#faf8f3",
            foreground="#111111",
            font=("Segoe UI", 10, "bold"),
        )
        style.configure("TNotebook", background="#faf8f3", borderwidth=0)
        style.configure("TNotebook.Tab", padding=(16, 8), font=("Segoe UI", 10))
        style.configure("Treeview", rowheight=28, font=("Segoe UI", 9),
background="#ffffff", fieldbackground="#ffffff")
        style.configure("Treeview.Heading", font=("Segoe UI", 9, "bold"))
        style.configure("Accent.TButton", font=("Segoe UI", 10, "bold"), padding=(14,
8))
        style.configure("TButton", font=("Segoe UI", 10), padding=(12, 8))
        style.configure("TSpinbox", padding=(6, 4))
        style.map("TButton", foreground=[("active", "#111111")])

    def _build_layout(self) -> None:
        header = ttk.Frame(self, padding=(20, 16, 20, 8))
        header.pack(fill="x")

        ttk.Label(header, text=APP_TITLE, style="Title.TLabel").pack(anchor="w")
        ttk.Label(header, text=MODELS_TEXT,
style="Subtitle.TLabel").pack(anchor="w", pady=(4, 0))

        plan_frame = ttk.LabelFrame(self, text="План зборок",
style="Panel.TLabelframe", padding=12)
        plan_frame.pack(fill="x", padx=20, pady=(4, 6))

        plan_row = ttk.Frame(plan_frame)
        plan_row.pack(fill="x")
        for idx, (code, label) in enumerate(
            [("P", "Mark4-Pro"), ("S", "Mark4-Standard"), ("L", "Mark4-Light")]
        ):
            cell = ttk.Frame(plan_row)
            cell.grid(row=0, column=idx, padx=(0 if idx == 0 else 18, 18), sticky="w")
            ttk.Label(cell, text=label, font=("Segoe UI", 10, "bold")).pack(anchor="w")
            spin = ttk.Spinbox(
                cell,
                from_=0,
                to=200,
```

```
                width=7,
                textvariable=self.plan_vars[code],
                font=("Segoe UI", 11),
                justify="center",
            )
            spin.pack(anchor="w", pady=(4, 0))
            self.locked_controls.append(spin)

        button_row = ttk.Frame(plan_frame)
        button_row.pack(fill="x", pady=(10, 4))
        self.start_button = ttk.Button(button_row, text="Запустити",
style="Accent.TButton", command=self.start_run)
        self.stop_button = ttk.Button(button_row, text="Стоп", command=self.stop_run)
        self.reset_button = ttk.Button(button_row, text="Скинути",
command=self.reset_all)
        self.clear_button = ttk.Button(button_row, text="Очистити журнал",
command=self.clear_log_only)
        self.refresh_button = ttk.Button(button_row, text="Оновити",
command=self.refresh)
        for widget in (self.start_button, self.stop_button, self.reset_button,
self.clear_button, self.refresh_button):
            widget.pack(side="left", padx=(0, 8))
        self.locked_controls.append(self.start_button)

        ttk.Label(
            plan_frame,
            textvariable=self.plan_summary_var,
            font=("Segoe UI", 10),
            foreground="#111111",
            wraplength=1180,
            justify="left",
        ).pack(anchor="w", pady=(4, 0))

        body = ttk.Frame(self)
        body.pack(fill="both", expand=True, padx=20, pady=(0, 8))
        body.columnconfigure(0, weight=1, uniform="main")
        body.columnconfigure(1, weight=1, uniform="main")
        body.rowconfigure(0, weight=1)
```

```
        requirements_frame = ttk.LabelFrame(
            body,
            text="Потрібно деталей",
            style="Panel.TLabelframe",
            padding=10,
        )
        requirements_frame.grid(row=0, column=0, sticky="nsew", padx=(0, 8))
        requirements_frame.columnconfigure(0, weight=1)
        requirements_frame.rowconfigure(0, weight=1)
        self.requirements_frame, self.requirements_table = self._make_treeview(
            requirements_frame,
            ["Шифр", "Назва", "Кількість"],
            height=18,
            columns_width={"Шифр": 150, "Назва": 300, "Кількість": 95},
        )
        self.requirements_frame.grid(row=0, column=0, sticky="nsew")

        journal_panel = ttk.LabelFrame(body, text="Журнал",
style="Panel.TLabelframe", padding=10)
        journal_panel.grid(row=0, column=1, sticky="nsew")
        journal_panel.columnconfigure(0, weight=1)
        journal_panel.rowconfigure(0, weight=3)
        journal_panel.rowconfigure(1, weight=2)

        self.journal_frame, self.journal_table = self._make_treeview(
            journal_panel,
            ["ID", "Крок", "Тип", "Шифр", "Модель", "Дільниця", "Повідомлення"],
            height=12,
            columns_width={
                "ID": 60,
                "Крок": 65,
                "Тип": 90,
                "Шифр": 105,
                "Модель": 70,
                "Дільниця": 75,
                "Повідомлення": 420,
            },
        )
        self.journal_frame.grid(row=0, column=0, sticky="nsew", pady=(0, 8))
```

```
        notebook = ttk.Notebook(journal_panel)
        notebook.grid(row=1, column=0, sticky="nsew")
        self._add_tab(notebook, "warehouse", "Склад", ["Шифр", "Назва", "Кількість"],
{"Назва": 320})
        self._add_tab(
            notebook,
            "buffers",
            "Дільниці",
            ["Дільниця", "Модель", "Шифр", "Назва", "Кількість"],
            {"Назва": 280},
        )
        self._add_tab(
            notebook,
            "finished",
            "Збірки",
            ["Модель", "План", "Зібрано", "Перевірено"],
            {"Модель": 230},
        )
        self._add_tab(
            notebook,
            "stats",
            "Статистика",
            ["Дільниця", "Модель", "Збірок", "Сумарний час", "Середній час"],
            {"Модель": 210},
        )

        ttk.Label(
            self,
            textvariable=self.status_var,
            padding=(20, 5, 20, 12),
            font=("Segoe UI", 10),
            foreground="#111111",
            background="#f4f1ea",
        ).pack(fill="x")

    def _make_treeview(self, parent, columns, height=10, columns_width=None):
        frame = ttk.Frame(parent)
        frame.columnconfigure(0, weight=1)
```

```
        frame.rowconfigure(0, weight=1)
        table = ttk.Treeview(frame, columns=columns, show="headings", height=height)
        y_scroll = ttk.Scrollbar(frame, orient="vertical", command=table.yview)
        x_scroll = ttk.Scrollbar(frame, orient="horizontal", command=table.xview)
        table.configure(yscrollcommand=y_scroll.set, xscrollcommand=x_scroll.set)
        table.grid(row=0, column=0, sticky="nsew")
        y_scroll.grid(row=0, column=1, sticky="ns")
        x_scroll.grid(row=1, column=0, sticky="ew")
        for column in columns:
            table.heading(column, text=column)
            width = 150
            if columns_width and column in columns_width:
                width = columns_width[column]
            table.column(column, width=width, minwidth=60, anchor="w")
        table.tag_configure("assembly", background="#d9f7d9")
        table.tag_configure("rework", background="#fff2cc")
        table.tag_configure("recycle", background="#ffd8b1")
        table.tag_configure("redirect", background="#dbeafe")
        table.tag_configure("shortage", background="#fde2e1")
        table.tag_configure("finished", background="#e7ecff")
        return frame, table

    def _add_tab(self, notebook: ttk.Notebook, key: str, title: str, columns: list[str],
width_map=None) -> None:
        frame, table = self._make_treeview(notebook, columns, height=7,
columns_width=width_map)
        self.tables[key] = table
        notebook.add(frame, text=title)

    def _bind_plan_updates(self) -> None:
        for var in self.plan_vars.values():
            var.trace_add("write", lambda *_: self._update_plan_preview())

    def _update_plan_preview(self) -> None:
        plan = self.get_plan()
        requirement_rows = self._calculate_required_parts(plan)
        total_drones = sum(plan.values())
        total_parts = sum(int(row[2]) for row in requirement_rows)
        self.plan_summary_var.set(
```

```
            "План: "
            f"Mark4-Pro {plan['P']}, Mark4-Standard {plan['S']}, Mark4-Light {plan['L']}.
"
            f"Загалом дронів: {total_drones}. "
            f"Загалом деталей за специфікацією: {total_parts}."
        )
        self._fill_static_table(self.requirements_table, requirement_rows)

    def _calculate_required_parts(self, plan: dict[str, int]) -> list[tuple[str, str, int]]:
        totals: dict[str, list[object]] = {}
        for part in PARTS:
            if part.model_code == "U":
                quantity = part.required_per_drone * sum(plan.values())
            else:
                quantity = part.required_per_drone * plan.get(part.model_code, 0)
            if quantity > 0:
                totals[part.code] = [part.code, part.name, quantity]
        return [tuple(values) for values in totals.values()]

    def get_plan(self) -> dict[str, int]:
        plan: dict[str, int] = {}
        for code, var in self.plan_vars.items():
            try:
                value = var.get()
            except tk.TclError:
                value = 0
            plan[code] = max(int(value), 0)
        return plan

    def start_run(self) -> None:
        if self.running:
            return
        plan = self.get_plan()
        if sum(plan.values()) <= 0:
            messagebox.showwarning("План порожній", "Вкажи хоча б одну збірку для
запуску.")
            return

        self.db.reset_for_plan(plan)
```

```
        self.simulator = ProductionSimulator(self.db)
        self.running = True
        self._set_controls_state("disabled")
        self.status_var.set("Симуляція запущена")
        self.refresh()
        self._schedule_step()

    def reset_all(self) -> None:
        if self.after_id:
            self.after_cancel(self.after_id)
            self.after_id = None
        self.running = False
        plan = self.get_plan()
        self.db.reset_for_plan(plan)
        self.db.clear_log()
        self.simulator = ProductionSimulator(self.db)
        self._set_controls_state("normal")
        self.status_var.set("План і журнал скинуто")
        self.refresh()

    def stop_run(self) -> None:
        if not self.running:
            self.status_var.set("Симуляція не запущена")
            return
        if self.after_id:
            self.after_cancel(self.after_id)
            self.after_id = None
        self.running = False
        if self.simulator is not None:
            self.simulator.stop_reason = "Зупинено користувачем"
        self._set_controls_state("normal")
        self.status_var.set("Симуляцію зупинено користувачем")
        self.refresh()

    def clear_log_only(self) -> None:
        self.db.clear_log()
        self.refresh()
        self.status_var.set("Журнал очищено")
```

```
def _set_controls_state(self, state: str) -> None:
    for widget in self.locked_controls:
        try:
            widget.configure(state=state)
        except tk.TclError:
            pass
    self.start_button.configure(state="normal" if state == "normal" else "disabled")
    self.stop_button.configure(state="normal")
    self.refresh_button.configure(state="normal")
    self.clear_button.configure(state="normal")

def _schedule_step(self) -> None:
    if not self.running or self.simulator is None:
        return
    if self.after_id:
        self.after_cancel(self.after_id)
    self.after_id = self.after(15, self._advance_run)

def _advance_run(self) -> None:
    if not self.running or self.simulator is None:
        return

    continue_run = self.simulator.step()
    self.refresh()

    if not continue_run:
        self.running = False
        self.after_id = None
        self._set_controls_state("normal")
        if self.simulator is not None:
            self.status_var.set(self.simulator.stop_reason or "Симуляцію завершено")
        self.refresh()
        self._show_final_report()
        return

    if self.simulator.tick >= 10000:
        self.running = False
        self.after_id = None
        self._set_controls_state("normal")
```

```
            self.status_var.set("Симуляція зупинена через ліміт кроків")
            self.refresh()
            messagebox.showwarning("Ліміт кроків", "Симуляція досягла
максимального ліміту кроків.")
            return

        self._schedule_step()

    def refresh(self) -> None:
        snap = self.db.snapshot()
        self._fill_table("warehouse", snap["warehouse"])
        self._fill_table("buffers", snap["buffers"])
        self._fill_table("finished", snap["finished"])
        self._fill_table("stats", snap["stats"])
        self._fill_table("events", snap["events"], event_table=True)
        if self.running and self.simulator is not None:
            self.status_var.set(
                f"Крок {self.simulator.tick}: "
                f"прийнято {self.simulator.summary.get('accepted', 0)}, "
                f"зібрано {self.simulator.summary.get('assembled', 0)}"
            )
        elif not self.status_var.get():
            self.status_var.set("Дані оновлено")
        if self.journal_table.get_children():
            self.journal_table.yview_moveto(1.0)

    def _fill_static_table(self, table, rows) -> None:
        table.delete(*table.get_children())
        for row in rows:
            table.insert("", "end", values=[str(value) for value in row])

    def _fill_table(self, key: str, rows, event_table: bool = False) -> None:
        table = self.journal_table if key == "events" else self.tables[key]
        table.delete(*table.get_children())
        if event_table:
            for row in rows:
                values = ["" if value is None else str(value) for value in list(row)]
                tags = self._event_tags(row["event_type"])
                item_id = table.insert("", "end", values=values, tags=tags)
```

```
            table.see(item_id)
        return
    for row in rows:
        values = ["" if value is None else str(value) for value in list(row)]
        table.insert("", "end", values=values)

def _event_tags(self, event_type: str) -> tuple[str, ...]:
    if event_type == "assembly_check":
        return ("assembly",)
    if event_type == "rework":
        return ("rework",)
    if event_type == "recycle":
        return ("recycle",)
    if event_type == "redirected":
        return ("redirect",)
    if event_type in {"shortage", "shortage_order"}:
        return ("shortage",)
    if event_type == "finished":
        return ("finished",)
    return ()

def _show_final_report(self) -> None:
    if self.simulator is None:
        return

    snap = self.db.snapshot()
    finished_rows = snap["finished"]
    stats_rows = snap["stats"]
    shortages = self.db.shortage_report()

    report = tk.Toplevel(self)
    report.title("Звіт про виконання плану")
    report.geometry("980x720")
    report.configure(bg="#f4f1ea")
    report.transient(self)
    report.grab_set()

    ttk.Label(report, text="Підсумковий звіт", font=("Segoe UI", 18,
"bold")).pack(anchor="w", padx=16, pady=(16, 6))
```

```
        summary = [
            f"Статус: {self.simulator.stop_reason or 'Невідомо'}",
            f"Витрачено кроків: {self.simulator.tick}",
            f"Прийнято деталей: {self.simulator.summary.get('accepted', 0)}",
            f"Доопрацювання: {self.simulator.summary.get('rework', 0)}",
            f"Переробка: {self.simulator.summary.get('recycle', 0)}",
            f"Перенаправлено: {self.simulator.summary.get('redirected', 0)}",
            f"Зібрано виробів: {self.simulator.summary.get('assembled', 0)}",
            f"Пройшли контроль якості: {self.simulator.summary.get('passed_quality',
0)}",
        ]
        ttk.Label(report, text="\n".join(summary), justify="left").pack(anchor="w",
padx=16, pady=(0, 12))

        models_frame = ttk.LabelFrame(report, text="Виконання плану", padding=10)
        models_frame.pack(fill="x", padx=16, pady=(0, 12))
        model_frame, model_table = self._make_treeview(
            models_frame,
            ["Модель", "План", "Зібрано", "Перевірено"],
            height=4,
            columns_width={"Модель": 250},
        )
        model_frame.pack(fill="x")
        model_map = {row["model_code"]: row for row in finished_rows}
        for code, name in [("P", "Mark4-Pro"), ("S", "Mark4-Standard"), ("L", "Mark4-
Light")]:
            row = model_map.get(code)
            model_table.insert(
                "",
                "end",
                values=[
                    name,
                    row["planned"] if row else 0,
                    row["assembled"] if row else 0,
                    row["passed_quality"] if row else 0,
                ],
            )

        times_frame = ttk.LabelFrame(report, text="Середній час складання по
```

```
дільницях", padding=10)
        times_frame.pack(fill="x", padx=16, pady=(0, 12))
        time_frame, time_table = self._make_treeview(
            times_frame,
            ["Дільниця", "Модель", "Збірок", "Сумарний час", "Середній час"],
            height=4,
            columns_width={"Модель": 240},
        )
        time_frame.pack(fill="x")
        for row in stats_rows:
            time_table.insert(
                "",
                "end",
                values=[
                    row["section"],
                    row["model"],
                    row["assemblies"],
                    row["total_assembly_time"],
                    row["average_time"],
                ],
            )

        shortages_frame = ttk.LabelFrame(report, text="Недостачі", padding=10)
        shortages_frame.pack(fill="both", expand=True, padx=16, pady=(0, 12))
        shortages_text = tk.Text(shortages_frame, height=10, wrap="word", font=("Segoe
UI", 10))
        shortages_text.pack(fill="both", expand=True)
        if shortages:
            for model_name, items in shortages.items():
                shortages_text.insert("end", f"{model_name}\n")
                for code, name, qty in items:
                    shortages_text.insert("end", f"  {code} - {name}: {qty} шт.\n")
                shortages_text.insert("end", "\n")
        else:
            shortages_text.insert("end", "Недостач не виявлено.")
        shortages_text.configure(state="disabled")

        if self.simulator.stop_reason != "План виконано":
            shortage_lines = []
```

```
            for model_name, items in shortages.items():
                parts_text = ", ".join(f"{code} ({qty} шт.)" for code, _name, qty in items)
                shortage_lines.append(
                    f"Кількість виробів {model_name} менша запланованої, замовлено недостатньо деталей: {parts_text}"
                )
            if shortage_lines:
                messagebox.showwarning("Недостача деталей", "\n".join(shortage_lines))
            else:
                messagebox.showwarning("Недостача деталей", "План не виконано, але точні недостачі не були сформовані.")

        ttk.Button(report, text="Закрити", command=report.destroy).pack(pady=(0, 16))


def run_app(db) -> None:
    app = PartsControlApp(db)
    app.mainloop()
```

database.py

```
from __future__ import annotations

import json
import sqlite3
from pathlib import Path
from typing import Iterable

from .catalog import MODELS, PARTS, MODEL_BY_CODE, PART_BY_CODE


class Database:
    def __init__(self, path: str) -> None:
        self.path = Path(path)
        self.path.parent.mkdir(parents=True, exist_ok=True)

    def connect(self) -> sqlite3.Connection:
        conn = sqlite3.connect(self.path)
        conn.row_factory = sqlite3.Row
        return conn
```

```
def initialize(self) -> None:
    with self.connect() as conn:
        conn.executescript(
            """
            CREATE TABLE IF NOT EXISTS models (
                code TEXT PRIMARY KEY,
                name TEXT NOT NULL,
                section INTEGER NOT NULL,
                strategic_priority INTEGER NOT NULL,
                target_units INTEGER NOT NULL
            );
            CREATE TABLE IF NOT EXISTS parts (
                code TEXT PRIMARY KEY,
                name TEXT NOT NULL,
                model_code TEXT NOT NULL,
                category TEXT NOT NULL,
                required_per_drone INTEGER NOT NULL,
                quality_rank INTEGER NOT NULL,
                acceptable_models TEXT NOT NULL
            );
            CREATE TABLE IF NOT EXISTS warehouse (
                part_code TEXT PRIMARY KEY REFERENCES parts(code),
                quantity INTEGER NOT NULL
            );
            CREATE TABLE IF NOT EXISTS section_buffer (
                section INTEGER NOT NULL,
                model_code TEXT NOT NULL,
                part_code TEXT NOT NULL REFERENCES parts(code),
                quantity INTEGER NOT NULL DEFAULT 0,
                PRIMARY KEY (section, part_code)
            );
            CREATE TABLE IF NOT EXISTS finished_goods (
                model_code TEXT PRIMARY KEY,
                planned INTEGER NOT NULL,
                assembled INTEGER NOT NULL DEFAULT 0,
                passed_quality INTEGER NOT NULL DEFAULT 0
            );
            CREATE TABLE IF NOT EXISTS assembly_stats (
```

```
                section INTEGER PRIMARY KEY,
                model_code TEXT NOT NULL,
                last_assembly_tick INTEGER NOT NULL DEFAULT 0,
                total_assembly_time INTEGER NOT NULL DEFAULT 0,
                assemblies INTEGER NOT NULL DEFAULT 0
            );
            CREATE TABLE IF NOT EXISTS event_log (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                tick INTEGER NOT NULL,
                event_type TEXT NOT NULL,
                part_code TEXT,
                target_model TEXT,
                target_section INTEGER,
                message TEXT NOT NULL
            );
            """
        )
        self._ensure_reference_data()

    def _ensure_reference_data(self) -> None:
        with self.connect() as conn:
            part_count = conn.execute("SELECT COUNT(*) FROM parts").fetchone()[0]
            if part_count == 0:
                return
            for model in MODELS:
                conn.execute(
                    """
                    INSERT OR IGNORE INTO assembly_stats(section, model_code,
last_assembly_tick, total_assembly_time, assemblies)
                    VALUES (?, ?, 0, 0, 0)
                    """,
                    (model.section, model.code),
                )

    def seed_if_empty(self, target_units: int = 50) -> None:
        with self.connect() as conn:
            count = conn.execute("SELECT COUNT(*) FROM parts").fetchone()[0]
            if count:
                return
```

```
        conn.executemany(
            "INSERT INTO models VALUES (?, ?, ?, ?, ?)",
            [(m.code, m.name, m.section, m.strategic_priority, target_units) for m in
MODELS],
        )
        conn.executemany(
            "INSERT INTO parts VALUES (?, ?, ?, ?, ?, ?, ?)",
            [
                (
                    p.code,
                    p.name,
                    p.model_code,
                    p.category,
                    p.required_per_drone,
                    p.quality_rank,
                    json.dumps(p.acceptable_models, ensure_ascii=False),
                )
                for p in PARTS
            ],
        )
        warehouse_rows = []
        for part in PARTS:
            if part.model_code == "U":
                quantity = part.required_per_drone * target_units * len(MODELS)
            else:
                quantity = part.required_per_drone * target_units
            warehouse_rows.append((part.code, quantity))
        conn.executemany("INSERT INTO warehouse VALUES (?, ?)",
warehouse_rows)

        buffer_rows = []
        for model in MODELS:
            for part in PARTS:
                if model.code in part.acceptable_models:
                    buffer_rows.append((model.section, model.code, part.code, 0))
        conn.executemany("INSERT OR IGNORE INTO section_buffer VALUES (?,
?, ?, ?)", buffer_rows)
        conn.executemany(
            "INSERT INTO finished_goods VALUES (?, ?, 0, 0)",
```

```
            [(m.code, target_units) for m in MODELS],
        )
        conn.executemany(
            "INSERT INTO assembly_stats VALUES (?, ?, 0, 0, 0)",
            [(m.section, m.code) for m in MODELS],
        )

    def reset_for_plan(self, plan: dict[str, int]) -> None:
        with self.connect() as conn:
            conn.execute("DELETE FROM event_log")
            conn.execute("DELETE FROM warehouse")
            conn.execute("DELETE FROM section_buffer")
            conn.execute("DELETE FROM finished_goods")
            conn.execute("DELETE FROM assembly_stats")
            conn.execute("UPDATE models SET target_units = CASE code WHEN 'P'
THEN ? WHEN 'S' THEN ? WHEN 'L' THEN ? ELSE target_units END",
                         (plan.get("P", 0), plan.get("S", 0), plan.get("L", 0)))

            warehouse_rows = []
            for part in PARTS:
                if part.model_code == "U":
                    quantity = part.required_per_drone * sum(plan.values())
                else:
                    quantity = part.required_per_drone * plan.get(part.model_code, 0)
                warehouse_rows.append((part.code, quantity))
            conn.executemany("INSERT INTO warehouse VALUES (?, ?)",
warehouse_rows)

            buffer_rows = []
            for model in MODELS:
                for part in PARTS:
                    if model.code in part.acceptable_models:
                        buffer_rows.append((model.section, model.code, part.code, 0))
            conn.executemany("INSERT OR IGNORE INTO section_buffer VALUES (?,
?, ?, ?)", buffer_rows)
            conn.executemany(
                "INSERT INTO finished_goods VALUES (?, ?, 0, 0)",
                [(m.code, plan.get(m.code, 0)) for m in MODELS],
            )
```

```
            conn.executemany(
                "INSERT INTO assembly_stats VALUES (?, ?, 0, 0, 0)",
                [(m.section, m.code) for m in MODELS],
            )

    def clear_log(self) -> None:
        with self.connect() as conn:
            conn.execute("DELETE FROM event_log")

    def fetch_all(self, query: str, params: Iterable[object] = ()) -> list[sqlite3.Row]:
        with self.connect() as conn:
            return list(conn.execute(query, tuple(params)))

    def fetch_one(self, query: str, params: Iterable[object] = ()) -> sqlite3.Row | None:
        with self.connect() as conn:
            return conn.execute(query, tuple(params)).fetchone()

    def move_from_warehouse(self, part_code: str) -> bool:
        with self.connect() as conn:
            row = conn.execute("SELECT quantity FROM warehouse WHERE part_code
= ?", (part_code,)).fetchone()
            if not row or row["quantity"] <= 0:
                return False
            conn.execute("UPDATE warehouse SET quantity = quantity - 1 WHERE
part_code = ?", (part_code,))
            return True

    def add_to_section(self, section: int, part_code: str, amount: int = 1) -> None:
        model_code = self.fetch_model_by_section(section)["code"]
        with self.connect() as conn:
            conn.execute(
                """
                INSERT INTO section_buffer(section, model_code, part_code, quantity)
                VALUES (?, ?, ?, ?)
                ON CONFLICT(section, part_code)
                DO UPDATE SET quantity = quantity + excluded.quantity
                """,
                (section, model_code, part_code, amount),
            )
```

```
    def update_buffer_quantity(self, section: int, part_code: str, quantity: int) -> None:
        with self.connect() as conn:
            conn.execute(
                "UPDATE section_buffer SET quantity = ? WHERE section = ? AND
part_code = ?",
                (quantity, section, part_code),
            )

    def mark_assembled(self, model_code: str, passed_quality: bool) -> None:
        with self.connect() as conn:
            conn.execute(
                "UPDATE finished_goods SET assembled = assembled + 1 WHERE
model_code = ?",
                (model_code,),
            )
            if passed_quality:
                conn.execute(
                    "UPDATE finished_goods SET passed_quality = passed_quality + 1
WHERE model_code = ?",
                    (model_code,),
                )

    def record_assembly_time(self, section: int, tick: int) -> int:
        with self.connect() as conn:
            row = conn.execute(
                "SELECT last_assembly_tick FROM assembly_stats WHERE section = ?",
                (section,),
            ).fetchone()
            previous_tick = row["last_assembly_tick"] if row else 0
            duration = max(tick - previous_tick, 1)
            conn.execute(
                """
                UPDATE assembly_stats
                SET last_assembly_tick = ?,
                    total_assembly_time = total_assembly_time + ?,
                    assemblies = assemblies + 1
                WHERE section = ?
                """,
```

```
            (tick, duration, section),
        )
        return duration

    def fetch_model_by_section(self, section: int) -> dict[str, object]:
        model = next(m for m in MODEL_BY_CODE.values() if m.section == section)
        return {"code": model.code, "name": model.name}

    def log(self, tick: int, event_type: str, message: str, part_code: str | None = None,
            target_model: str | None = None, target_section: int | None = None) -> None:
        with self.connect() as conn:
            conn.execute(
                """
                INSERT INTO event_log(tick, event_type, part_code, target_model,
target_section, message)
                VALUES (?, ?, ?, ?, ?, ?)
                """,
                (tick, event_type, part_code, target_model, target_section, message),
            )

    def snapshot(self) -> dict[str, list[sqlite3.Row]]:
        return {
            "warehouse": self.fetch_all(
                """
                SELECT p.code, p.name, w.quantity
                FROM warehouse w JOIN parts p ON p.code = w.part_code
                ORDER BY p.code
                """
            ),
            "buffers": self.fetch_all(
                """
                SELECT b.section, m.name AS model, p.code, p.name, b.quantity
                FROM section_buffer b
                JOIN parts p ON p.code = b.part_code
                JOIN models m ON m.code = b.model_code
                WHERE b.quantity > 0
                ORDER BY b.section, p.code
                """
            ),
```

```
        "finished": self.fetch_all("SELECT * FROM finished_goods ORDER BY
model_code"),
        "stats": self.fetch_all(
            """
            SELECT a.section, m.name AS model, a.assemblies, a.total_assembly_time,
                CASE WHEN a.assemblies = 0 THEN 0
                    ELSE ROUND(CAST(a.total_assembly_time AS REAL) /
a.assemblies, 2)
                END AS average_time
            FROM assembly_stats a
            JOIN models m ON m.code = a.model_code
            ORDER BY a.section
            """
        ),
        "events": self.fetch_all("SELECT * FROM event_log ORDER BY id DESC
LIMIT 80"),
    }

def plan_completed(self) -> bool:
    rows = self.fetch_all("SELECT planned, assembled FROM finished_goods")
    return bool(rows) and all(row["assembled"] >= row["planned"] for row in rows)

def warehouse_empty(self) -> bool:
    row = self.fetch_one("SELECT SUM(quantity) AS total FROM warehouse")
    return row is None or (row["total"] or 0) <= 0

def shortage_report(self) -> dict[str, list[tuple[str, str, int]]]:
    result: dict[str, list[tuple[str, str, int]]] = {}
    finished = self.fetch_all("SELECT * FROM finished_goods")
    for goods in finished:
        remaining_units = max(goods["planned"] - goods["assembled"], 0)
        if remaining_units == 0:
            continue
        model = MODEL_BY_CODE[goods["model_code"]]
        missing: list[tuple[str, str, int]] = []
        requirements = [p for p in PARTS if p.model_code in ("U", model.code)]
        for part in requirements:
            required = part.required_per_drone * remaining_units
            buffer_qty = self._buffer_available_for_requirement(model.section, part)
```

```
                warehouse_qty = self._warehouse_quantity(part.code)
                deficit = max(required - buffer_qty - warehouse_qty, 0)
                if deficit:
                    missing.append((part.code, part.name, deficit))
            if missing:
                result[model.name] = missing
        return result

    def _warehouse_quantity(self, part_code: str) -> int:
        row = self.fetch_one("SELECT quantity FROM warehouse WHERE part_code =
?", (part_code,))
        return int(row["quantity"]) if row else 0

    def _buffer_available_for_requirement(self, section: int, required_part) -> int:
        rows = self.fetch_all(
            """
            SELECT b.part_code, b.quantity
            FROM section_buffer b
            JOIN parts p ON p.code = b.part_code
            WHERE b.section = ? AND b.quantity > 0
            """,
            (section,),
        )
        quantity = 0
        for row in rows:
            candidate = PART_BY_CODE[row["part_code"]]
            if required_part.model_code == "U":
                if candidate.code == required_part.code:
                    quantity += row["quantity"]
            elif candidate.code == required_part.code:
                quantity += row["quantity"]
            elif candidate.model_code != "U" and candidate.category ==
required_part.category and candidate.quality_rank >= required_part.quality_rank:
                quantity += row["quantity"]
        return quantity
```

catalog.py

```
from __future__ import annotations
```

```
from dataclasses import dataclass


@dataclass(frozen=True)
class DroneModel:
    code: str
    name: str
    section: int
    strategic_priority: int


@dataclass(frozen=True)
class PartSpec:
    code: str
    name: str
    model_code: str
    category: str
    required_per_drone: int
    quality_rank: int
    acceptable_models: tuple[str, ...]


MODELS: tuple[DroneModel, ...] = (
    DroneModel("P", "Mark4-Pro", 1, 3),
    DroneModel("S", "Mark4-Standard", 2, 2),
    DroneModel("L", "Mark4-Light", 3, 1),
)


UNIVERSAL_MODELS = ("P", "S", "L")


PARTS: tuple[PartSpec, ...] = (
    PartSpec("U-HD-001", "Гвинт M3x6 мм", "U", "hardware", 16, 1,
UNIVERSAL_MODELS),
    PartSpec("U-HD-002", "Гвинт M3x14 мм", "U", "hardware", 8, 1,
UNIVERSAL_MODELS),
    PartSpec("U-RC-001", "Радіоприймач ELRS 2.4GHz", "U", "radio", 1, 1,
UNIVERSAL_MODELS),
```

```
    PartSpec("U-OT-001", "Ремінець акумулятора Velcro", "U", "other", 1, 1,
UNIVERSAL_MODELS),
    PartSpec("U-EL-001", "Конденсатор живлення Low ESR", "U", "electronics", 1,
1, UNIVERSAL_MODELS),
    PartSpec("U-EL-002", "Кабель живлення з роз'ємом XT60", "U", "electronics", 1,
1, UNIVERSAL_MODELS),
    PartSpec("L-RM-001", "Рама Mark4-Light, полегшена", "L", "frame", 1, 1, ("L",)),
    PartSpec("L-EL-001", "Польотний стек FC+ESC, економ-клас", "L",
"electronics", 1, 1, ("L",)),
    PartSpec("L-MT-001", "Двигуни, базовий клас", "L", "motor", 4, 1, ("L",)),
    PartSpec("L-PR-001", "Пропелери, стандартні", "L", "propeller", 4, 1, ("L",)),
    PartSpec("L-VC-001", "Відеосистема аналогова, базова", "L", "video", 1, 1,
("L",)),
    PartSpec("S-RM-001", "Рама Mark4-Standard", "S", "frame", 1, 2, ("S", "L")),
    PartSpec("S-EL-001", "Польотний контролер FC, стандарт", "S", "electronics", 1,
2, ("S", "L")),
    PartSpec("S-EL-002", "Регулятор швидкості ESC, стандарт", "S", "electronics", 1,
2, ("S", "L")),
    PartSpec("S-MT-001", "Двигуни, стандартний клас", "S", "motor", 4, 2, ("S",
"L")),
    PartSpec("S-PR-001", "Пропелери, посилені", "S", "propeller", 4, 2, ("S", "L")),
    PartSpec("S-VC-001", "Відеосистема аналогова, середній клас", "S", "video", 1,
2, ("S", "L")),
    PartSpec("S-OT-001", "Захист променів, 3D-друк", "S", "other", 1, 2, ("S", "L")),
    PartSpec("P-RM-001", "Рама Mark4-Pro, преміум", "P", "frame", 1, 3, ("P", "S",
"L")),
    PartSpec("P-EL-001", "Польотний контролер, преміум", "P", "electronics", 1, 3,
("P", "S", "L")),
    PartSpec("P-EL-002", "Регулятор швидкості ESC, преміум", "P", "electronics", 1,
3, ("P", "S", "L")),
    PartSpec("P-MT-001", "Двигуни преміум, низька вібрація", "P", "motor", 4, 3,
("P", "S", "L")),
    PartSpec("P-PR-001", "Пропелери карбонові", "P", "propeller", 4, 3, ("P", "S",
"L")),
    PartSpec("P-VC-001", "Відеосистема цифрова HD", "P", "video", 1, 3, ("P", "S",
"L")),
    PartSpec("P-OT-001", "Захист корпусу та моторів", "P", "other", 1, 3, ("P", "S",
"L")),
    PartSpec("P-OT-002", "Модуль GPS + компас", "P", "navigation", 1, 3, ("P", "S",
```

```
"L")),
)


PART_BY_CODE = {part.code: part for part in PARTS}
MODEL_BY_CODE = {model.code: model for model in MODELS}
```

init.py

```
__all__ = [
    "catalog",
    "database",
    "priority",
    "simulator",
    "gui",
]
```