МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Методичні вказівки до лабораторних занять
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Професійна освіта (комп'ютерні технології)
галузі знань А Освіта
спеціальності А5.39 Професійна освіта (Цифрові технології)
денної та заочної форм навчання

Луцьк 2026

УДК 004.4:378.147
М-48
До друку

Голова вченої ради факультету цифрових, освітніх та соціальних технологій Луцького національного технічного університету _____ Г.А. Герасимчук

Затверджено вченою радою факультету цифрових, освітніх та соціальних технологій Луцького національного технічного університету,
протокол № ____ від «___»_____ 2026 року.

Електронна копія друкованого видання передана для внесення в репозитарій Луцького національного технічного університету.
Директор бібліотеки _____ Н.П. Поліщук

Рекомендовано до видання на засіданні кафедри цифрових освітніх технологій Луцького національного технічного університету,
протокол № _____ від «___»_____ 2026 року.
Завідувач кафедри цифрових освітніх технологій _____ В.В. Кабак

Укладач: _____ Ю.Є. Мельничук, кандидат педагогічних наук, доцент кафедри цифрових освітніх технологій ЛНТУ.

Рецензент: _____ П.В. Саварин, кандидат педагогічних наук, доц. кафедри ЦОТ ЛНТУ.

Відповідальний за випуск: _____ В.В. Кабак, кандидат педагогічних наук, доцент кафедри ЦОТ, завідувач кафедри ЦОТ ЛНТУ.

М-48 **Проектування інформаційних систем** [Текст]: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання/ уклад. Ю.Є. Мельничук Луцьк: ЛНТУ, 2026. 60 с.

Методичне видання складене відповідно до діючої програми курсу «Проектування інформаційних систем» з метою надання методичної допомоги у процесі вивчення курсу. Видання містить теоретичний матеріал з дисципліни «Проектування інформаційних систем».

Зміст

ВСТУП

Сучасний розвиток інформаційних технологій зумовлює зростання ролі інформаційних систем у різних сферах діяльності суспільства. Інформаційні системи використовуються для автоматизації управління, обробки даних, підтримки прийняття рішень, організації бізнес-процесів та забезпечення ефективної взаємодії між користувачами і програмними засобами. У зв'язку з цим особливого значення набуває підготовка фахівців, здатних здійснювати проєктування, розробку та впровадження сучасних інформаційних систем.

Дисципліна «Проєктування інформаційних систем» спрямована на формування у здобувачів вищої освіти теоретичних знань та практичних умінь щодо аналізу предметної області, моделювання інформаційних процесів, розроблення структури інформаційних систем та застосування сучасних методів і засобів їх проєктування.

Важливу роль у засвоєнні навчального матеріалу відіграють лабораторні заняття, під час яких здобувачі освіти мають можливість застосовувати отримані теоретичні знання на практиці, виконувати завдання з аналізу та моделювання інформаційних систем, а також формувати навички роботи з сучасними інструментами проєктування.

Метою даних методичних вказівок є надання здобувачам вищої освіти методичної допомоги під час виконання лабораторних робіт з дисципліни «Проєктування інформаційних систем». У методичних вказівках подано опис лабораторних робіт, рекомендації щодо їх виконання, контрольні запитання та практичні завдання, спрямовані на закріплення теоретичних знань та розвиток практичних навичок проєктування інформаційних систем.

Методичні вказівки призначені для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Професійна освіта (Комп'ютерні технології)» галузі знань А Освіта, спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання.

Лабораторна робота № 1. Моделювання систем. Процеси керування

Мета: Ознайомитися зі способами моделювання систем. Навчитися моделювати системи.

Теоретичні відомості

У процесі формалізації вимог до системи та на етапі проектування система оглядається як сукупність компонентів і взаємозв'язків між ними. Для цього використовуються моделі системної архітектури, які в графічному виді надають всю організацію системи, тобто її компоненти й взаємозв'язки між ними.

Архітектура системи звичайно представляється у вигляді блокової діаграми (блок-схеми), де блоки відповідають основним підсистемам, а існуючі зв'язки між підсистемами позначаються лініями зі стрілками, що з'єднують окремі блоки діаграми. Зв'язку можуть відповідати потокам даних, послідовності включення підсистем у роботу або які-небудь інші типи залежності.

На рис. 1.1 представлена блок-схема основних компонентів системи сигналізації, що попереджає про несанкціоноване проникнення в житло. У табл. 1 наведений короткий опис підсистем, яким відповідають певні блоки на рис. 1.

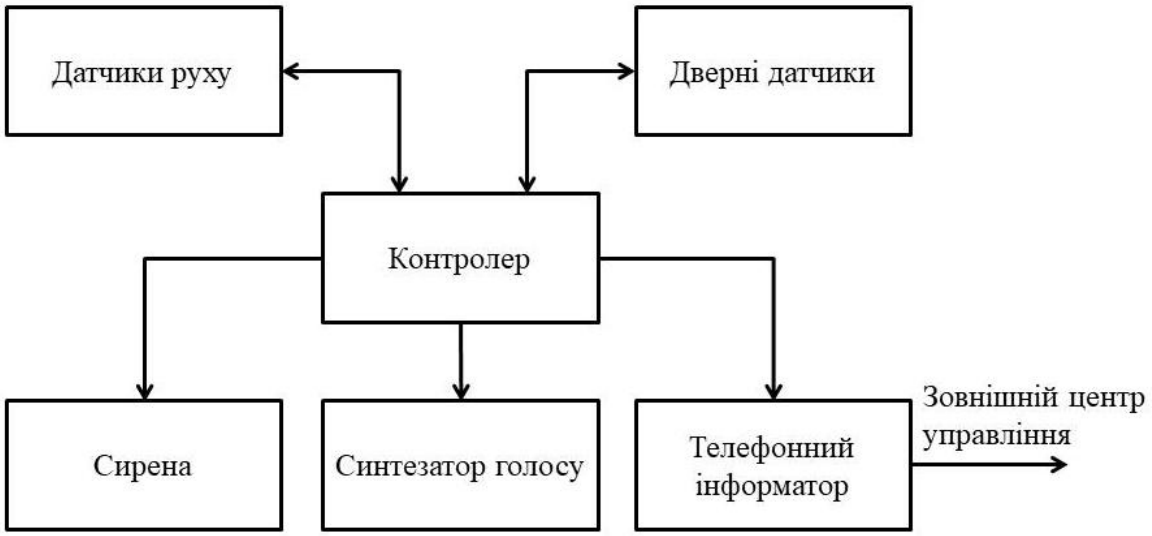

Рисунок 1.1 – Блок-схема основних компонентів системи сигналізації

На цьому рівні деталізації система розбивається на окремі підсистеми. Кожна підсистема, у свою чергу, може бути представлена як декомпозиція своїх функціональних компонентів. Це такі компоненти підсистеми, які, виходячи із призначення підсистеми, виконують яку-небудь одну функцію. На противагу цьому підсистема зазвичай виконує кілька функцій. Звичайно, декомпозицію підсистем (і самої системи) можна проводити за іншими ознаками, наприклад, конструктивними чи технологічними.

Таблиця 1.1 – Функціональні підсистеми системи сигналізації

Підсистема	**Опис**
Датчики руху	Реагують на рух у кімнатах, які контролює система
Дверні датчики	Визначають, чи відкриті зовнішні двері будинку
Контролер	Управляє діями всієї системи
Сирена	Видає потужний, звуковий сигнал при незаконному проникненні в житло
Синтезатор голосу	Синтезує голосове повідомлення про проникнення в будинок
Телефонний інформатор	Робить зовнішній телефонний дзвінок для повідомлення служби безпеки (наприклад, поліції) про проникнення в будинок

Історично склалося так, що модель системної архітектури використовується для вичленовування апаратних і програмних компонентів системи, які зазвичай розробляються паралельно. Разом з тим протиставлення "апаратні засоби — програмне забезпечення" у сучасних системах найчастіше недоречне й несуттєве. Оскільки практично всі системні компоненти мають певні обчислювальні можливості.

На рівні системної архітектури більш раціонально класифікувати підсистеми відповідно до виконуваних ними функцій, не акцентуючи спеціально увагу на тому, є вони апаратними чи програмними компонентами. Питання про те, чи буде дана функція реалізована апаратно або програмно, часто вирішується на основі нетехнічних факторів, таких як час, необхідний для створення компонента, або виходячи з наявності на ринку промислових виробів, що підходять для готових пристроїв.

Блок-схеми можна використовувати для подання систем будь-якого розміру. На рис. 2 показана архітектура значно складнішої системи керування польотами. Ця система містить кілька основних підсистем, які самі є системами великого розміру. Напрямок інформаційних потоків між підсистемами показано з'єднуючими їх лініями зі стрілками.

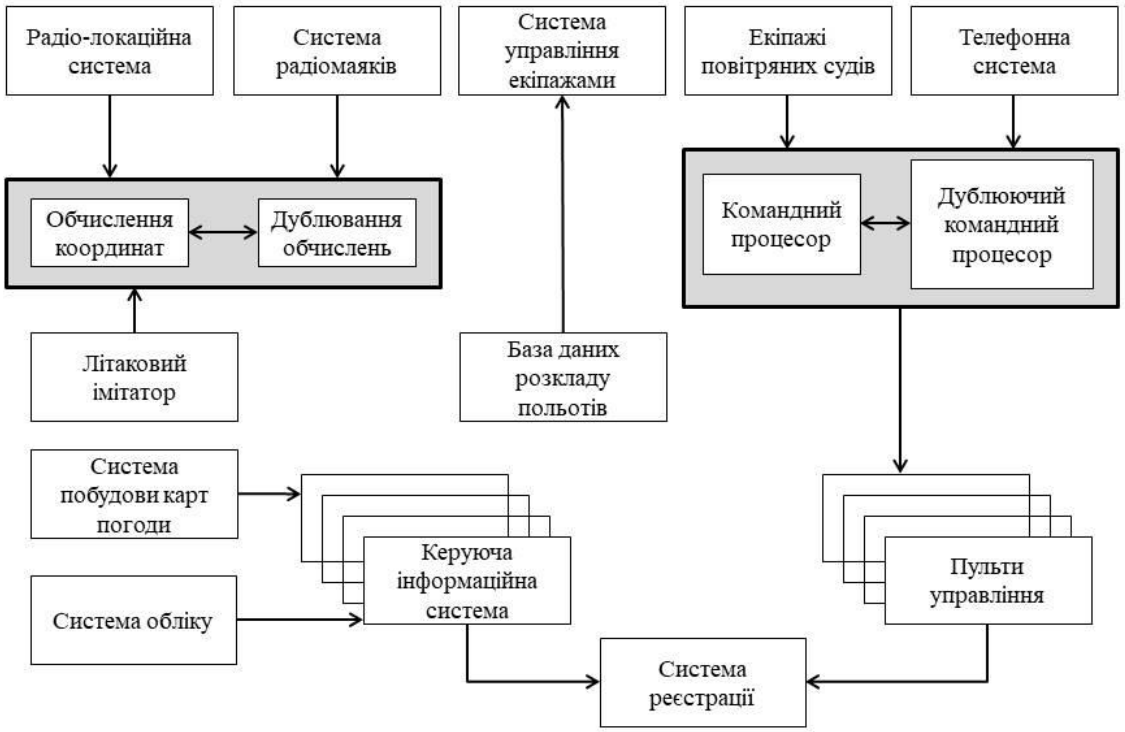

Рисунок 1.2 – Архітектура системи управління польотами

Неможливо описати й стандартизувати всі роботи, виконувані проектом по створенню ПЗ. Ці роботи досить істотно залежать від організації, де розробляється ПЗ, і від типу створюваного програмного продукту. Але в будь-якому випадку більшість менеджерів відповідальні за виконання всіх або деяких з наведених нижче процесів керування:

- написання пропозицій по створенню ПЗ;
- планування й складання графіка робіт зі створення ПЗ;
- оцінювання вартості проекту;
- контроль за ходом виконання робіт;
- підбір персоналу;
- написання звітів і представлень.

Перша стадія програмного проекту може складатися з написання пропозицій по реалізації цього проекту. Пропозиції повинні містити опис цілей проектів і способів їх досягнення. Вони також зазвичай містять у собі оцінки фінансових і часових витрат на виконання проекту.

Написання пропозицій – дуже відповідальна робота, тому що для багатьох організацій питання про те, чи буде проект виконуватися самою організацією або розроблятися за контрактом сторонньою компанією, є критичним.

На етапі планування проекту визначаються процеси, етапи й отримані на кожному з них результати, які повинні привести до виконання проекту. Реалізація цього плану приведе до досягнення цілей проекту. Визначення вартості проекту

прямо пов'язане з його плануванням, оскільки тут оцінюються ресурси, що вимагаються для виконання плану.

Контроль за ходом виконання робіт (моніторинг проекту) – це безперервний процес, що триває протягом усього терміну реалізації проекту. Менеджер повинен постійно відслідковувати хід реалізації проекту й порівнювати фактичні й нові показники виконання робіт з їхньою вартістю. Хоча багато організацій мають механізми формального моніторингу робіт, досвідчений менеджер може скласти ясну картину про стадію розвитку проекту просто шляхом неформального спілкування з розробником.

Неформальний моніторинг часто допомагає виявити потенційні проблеми, які в явному виді можуть виявитися пізніше. Наприклад, щоденне обговорення ходу виконання робіт може виявити окремі недоробки в створюваному програмному продукті. Замість очікування звітів, у яких буде відбитий факт "пробуксовки" графіка робіт, менеджер може обговорити з фахівцями намічені програмістські проблеми й не допустити зриву графіка робіт.

Протягом реалізації проекту звичайно відбувається кілька формальних контрольних перевірок ходу виконання робіт зі створення ПЗ. Такі перевірки повинні дати загальну картину ходу реалізації проекту в цілому й показати, наскільки вже розроблена частина ПЗ відповідає цілям проекту.

Час виконання великих програмних проектів може займати кілька років. Протягом цього часу цілі й наміри організації, що замовила програмний проект, можуть істотно змінитися. Може виявитися, що розроблювальний програмний проект уже непотрібен або вихідні вимоги до створюваного ПЗ просто застаріли і їх необхідно кардинально змінювати. У такій ситуації керівництво організації-розробника може застосовувати рішення про припинення розробки ПЗ або про зміну проекту в цілому, враховуючи змінені цілі й наміри організації-замовника.

Хід роботи

1. Ознайомитись із теоретичними відомостями за темою лабораторної роботи.
2. Сформулювати пропозиції щодо створення програмного забезпечення відповідно до власного варіанту.
3. Розробити архітектуру системи та зобразити її у вигляді схеми (відповідно до власного варіанту).
4. Сформулювати висновки щодо важливості проробленої роботи у процесі проектування обраної програмної системи (відповідно до власного варіанту).
5. Оформити звіт із лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Варіанти: (варіант відповідає номеру студента у журналі**)**

1. Інформаційна система ВНЗ
2. Інформаційна система торгової організації
3. Інформаційна система медицинських організацій міста
4. Інформаційна система автопідприємства міста
5. Інформаційна система проектної організації
6. Інформаційна система авіабудівного підприємства
7. Інформаційна система військового округу
8. Інформаційна система будівельної організації
9. Інформаційна система бібліотечного фонду міста
10. Інформаційна система спортивних організацій міста
11. Інформаційна система автомобілебудівного підприємства
12. Інформаційна система готельного підприємства
13. Інформаційна система магазину автозапчастин
14. Інформаційна система аптеки
15. Інформаційна система туристичного клубу.

Контрольні запитання

1. Що таке моделювання систем і з якою метою воно використовується під час проєктування інформаційних систем?
2. Які основні типи моделей застосовуються при моделюванні систем?
3. Що таке система керування та які її основні компоненти?
4. Які етапи включає процес моделювання системи?
5. Яку роль відіграє моделювання у процесі аналізу та проєктування інформаційних систем?
6. У чому полягає взаємозв'язок між моделлю системи та реальним об'єктом?

Лабораторна робота №2. Порівняння типів моделей життєвого циклу

Мета: Виконати порівняльний аналіз різних типів життєвого циклу

Теоретичні відомості

Розглянуті підходи щодо побудови різних видів моделей ЖЦ базуються на процесному підході до виконання програмних проектів. Вони використовувалися на практиці під час створення різних типів моделей ЖЦ, до яких належать такі моделі: каскадна, спіральна, інкрементна, еволюційна та ін.

Каскадна модель

Однією з перших почала застосовуватися каскадна модель, де кожна робота виконується один раз і в певному порядку.

Тобто вважається, що кожна робота має бути виконана настільки ретельно, що після її закінчення і переходу до наступного етапу, повертатися до попереднього не буде потреби. Розробник перевіряє проміжний результат відомими методами верифікації і фіксує його як готовий еталон для наступного процесу.

Згідно з даною моделлю роботи і завдання процесу розроблення зазвичай виконуються послідовно.

Проте допоміжні і організаційні процеси (контроль вимог, керування якістю і ін.), як правило, виконуються разом з процесами розробки ПЗ. У даній моделі повернення до початкового процесу передбачається після супроводження і виправлення помилок.

Особливість такої моделі полягає у фіксації послідовних процесів розроблення програмного продукту. В її основу покладена модель фабрики, де продукт проходить стадії від задуму до виробництва, потім його передають замовнику у вигляді готового виробу, де заміна не передбачена, хоча можна подати інший подібний виріб. Недоліки цієї моделі такі:

– процес створення ПС не завжди вкладається в таку жорстку форму і послідовність дій;

– не враховуються змінювані потреби користувачів, нестабільні умови зовнішнього середовища, які впливають на зміни вимог до ПС під час її розроблення;

– значний розрив між часом внесення помилки (наприклад, на процесі проектування) і часом її виявлення (при супроводі), що призводить до суттєвої переробки ПС.

При застосуванні каскадної моделі можуть спостерігатися такі чинники ризику:

– вимоги до ПС недостатньо чітко сформульовані, або не враховують перспективи розвитку ОС, середовищ і т.п.;

– громіздка система, що не допускає компонентної декомпозиції, може викликати проблеми щодо розміщення її в пам'яті або на платформах, не передбачених у вимогах;

– внесення швидких змін до технології і у вимоги може погіршити процес розроблення окремих частин системи або системи в цілому;

– обмеження на ресурси (людські, програмні, технічні і ін.) в ході розробки можуть звузити окремі можливості реалізації системи;

– отриманий продукт може виявитися не придатним для застосування внаслідок нерозуміння розробниками вимог або функцій системи або недостатньо проведеного тестування.

Переваги реалізації системи за допомогою каскадної моделі такі:

– всі завдання підсистем і системи реалізуються одночасно, завдяки чому не можна забути жодного завдання, а це сприяє встановленню стабільних зв'язків між ними;

– повністю розроблену систему з документацією на неї легше супроводжувати, тестувати, фіксувати помилки і вносити зміни не хаотично, а цілеспрямовано, починаючи з вимог, наприклад, додавати або замінювати деякі функції і повторювати процес.

Каскадну модель можна розглядати як модель ЖЦ, придатну для створення першої версії ПЗ з метою перевірки реалізованих в ній функцій. При супроводі і експлуатації можуть бути виявлені різного роду помилки, виправлення яких спричинить повторне виконання всіх процесів, починаючи з уточнення вимог.

Інкрементна модель

Цю модель (incremental) ще називають моделлю з нарощуванням або з приростом. Її суть полягає в розробці продукту ітераціями, і кожна ітерація закінчується випуском працездатної версії. У кожній новій версії додаються деякі функціональні можливості. Розробка системи починається з визначення набору всіх вимог до ПС і ділення процесу розроблення на ітерації. Кожна ітерація реалізується послідовно з використанням процесів ЖЦ і фіксації робочої версії системи, системи, що поступово наближається до остаточної версії.

Перша проміжна версія системи, що створюється (випуск 1), реалізує частину вимог, у подальшу версію (випуск 2) додають додаткові вимоги до тих пір, поки не будуть остаточно виконані всі вимоги і розв’язані задачі розробки системи.

Для кожної проміжної версії на процесах ЖЦ виконуються необхідні процеси, роботи і завдання, зокрема, аналіз вимог і створення нової архітектури, які можуть бути виконані одночасно.

Процеси розроблення технічного проекту програмної системи, її програмування і тестування, збирання і кваліфікаційні випробування ПС виконуються при створенні кожної подальшої версії.

Відповідно до даної моделі ЖЦ, процеси якої практично такі самі, як і в каскадній моделі, наголос робиться на побудову деякої закінченої проміжної версії, а завдання процесу розроблення виконуються послідовно або частково паралельно для ряду окремих проміжних структур версії.

Роботи і завдання процесу розроблення наступної версії системи з додатковими вимогами або функціями можуть виконуватися неодноразово в одній тій же послідовності для всіх проміжних версій системи. Процеси супроводження і експлуатації можуть бути реалізовані разом з процесом розроблення версії шляхом перевірки частково реалізованих вимог в кожній проміжній версії і так до отримання кінцевого варіанта системи. Допоміжні і організаційні процеси ЖЦ зазвичай виконуються разом з процесом розроблення версії системи і до кінця розробки збираються дані, на підставі яких можна встановити рівень завершеності і якості виготовленої системи.

При застосуванні даної моделі необхідно враховувати такі чинники ризику:

– вимоги складені з урахуванням можливості їх зміни при реалізації продукту;

– всі можливості системи потрібно реалізувати одразу;

– швидка зміна технології і вимог до системи може призвести до порушення отриманої структури системи;

– обмеження в ресурсному забезпеченні (виконавці, фінанси) можуть призвести до несвоєчасного введення системи в експлуатацію.

Дану модель ЖЦ доцільно використовувати, у випадках, коли:

– бажано реалізувати деякі можливості системи швидко за рахунок створення проміжної версії продукту;

– система декомпозується на окремі складові частини, які можна реалізовувати як деякі самостійні проміжні або готові продукти;

– можливе збільшення фінансування на розробку окремих частин системи.

Спіральна модель

Виходячи з можливості внесення змін, як в процес, так і в проміжний продукт було створено спіральну модель ЖЦ.

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації не відповідають дійсному стану розробки.

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку правильності та прийняття рішення про перехід на

наступний виток або повернення на попередній виток для доопрацювання на ньому проміжного продукту.

Відмінність цієї моделі від каскадної полягає в можливості багато разів повертатися до процесу формулювання вимог і до повторної розробки версії системи з будь-якого процесу моделі.

Для програмного продукту така модель не дуже підходить з декількох причин. По-перше, висловлення вимог замовником носить суб'єктивний характер, вимоги можуть багаторазово уточнюватися протягом розробки ПС і навіть після завершення та випробовування, і часом може з'ясуватися, що замовник «хотів зовсім інше». По-друге, змінюються обставини та умови використання системи, тому загальновизнаним законом програмної інженерії є закон еволюції, який сформулюємо так: кожна діюча ПС з часом потребує внесення змін або виводиться з експлуатації.

При необхідності внесення змін до системи на кожному витку з метою отримання нової версії системи обов'язково вносяться зміни в заздалегідь зафіксовані вимоги, після чого повертаються на попередній виток спіралі для продовження реалізації нової версії системи з урахуванням усіх змін.

Еволюційна модель

У разі еволюційної моделі система послідовно розробляється з блоків конструкцій. На відміну від інкрементної моделі в еволюційній моделі вимоги встановлюються частково і уточнюються в кожному наступному проміжному блоці структури системи.

Використання еволюційної моделі припускає проведення дослідження предметної області для вивчення потреб її замовника і аналізу можливості застосування цієї моделі для реалізації. Модель використовується для розробки нескладних і некритичних систем, де головною вимогою є реалізація функцій системи. При цьому вимоги не можуть бути визначені відразу і повністю. Тому розробка системи здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи–прототипу, на якому перевіряється реалізація вимог. Іншими словами, такий процес за своєю суттю є ітераційним, з етапами розробки, що повторюються, починаючи від змінених вимог і закінчуючи отриманням готового продукту. В деякому розумінні до цього типу моделі можна віднести спіральну модель.

Розвитком цієї моделі є модель еволюційного прототипування в рамках усього ЖЦ розробки ПС.

У літературі вона часто називається моделлю швидкої розробки програм RAD (Rapid Application Development). У даній моделі наведені дії, які пов'язані з аналізом її застосовності для конкретного виду системи, а також

обстеженням замовника для визначення потреб користувача при розробці плану створення прототипу.

У моделі є дві головні ітерації розробки функціонального прототипу, проектування і реалізації системи з метою перевірки, чи задовольняє вона всі функціональні і нефункціональні вимоги. Основною ідеєю цієї моделі є моделювання окремих функцій системи в прототипі і поступове еволюційне його доопрацювання до виконання всіх заданих функціональних вимог.

Ітерацій з отримання проміжних варіантів прототипу може бути декілька, в кожній з яких додається функція і повторно моделюється робота прототипу. І так до тих пір, поки не будуть промодельовані всі функції, задані у вимогах до системи. Після цього виконується ще одна ітерація – остаточне програмування для отримання готової системи.

Ця модель застосовується для систем, в яких найбільш важливими є функціональні можливості, і які необхідно швидко продемонструвати на CASE-засобах.

Оскільки проміжні прототипи системи відповідають реалізації деяких функціональних вимог, їх можна перевіряти і під час супроводу і експлуатації, тобто разом з процесом розробки чергових прототипів системи. При цьому допоміжні і організаційні процеси можуть виконуватися разом з процесом розроблення і накопичувати відомості за даними кількісних і якісних оцінок на процесах розроблення.

При цьому враховуються такі чинники ризику:

– реалізація всіх функцій системи одночасно може призвести до громіздкості;

– обмежені людські ресурси зайняті розробкою протягом тривалого часу.

Модель розвивається у напряму додавання нефункціональних вимог до системи, пов'язаних із захистом і безпекою даних, несанкціонованим доступом до них і ін.

Хід роботи

1. Опираючись на теоретичний опис, схематично зобразити принцип дії кожної з моделей. Схеми помістити у звіт.
2. Що таке RAD? Визначити переваги, описати їх у звіті.
3. Визначити, яку з моделей найдоцільніше використати при розробці програмного продукту із попередньої лабораторної роботи. Обгрунтуйте свій вибір.
4. Сформулювати висновки щодо ефективності застосування кожної з моделей.
5. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке життєвий цикл програмного забезпечення?
2. Які основні моделі життєвого циклу програмного забезпечення вам відомі?
3. У чому особливості каскадної моделі життєвого циклу?
4. Які переваги та недоліки спіральної моделі?
5. Чим відрізняється ітераційна модель від інкрементної?
6. У яких випадках доцільно використовувати гнучкі (Agile) методології?

Лабораторна робота №3. Керування ризиками. Функціональні й нефункціональні вимоги

Мета: Ознайомитися з можливими ризиками програмних проектів. Навчитися визначати й аналізувати ризики. Ознайомитися з функціональними й нефункціональні вимогами. Навчиться розробляти функціональні й нефункціональні вимоги.

Теоретичні відомості

Важливою частиною роботи менеджера проекту є оцінка ризиків, які можуть вплинути на графік робіт або на якість створюваного програмного *продукту,* і розробка заходів щодо запобігання ризиків. Результати аналізу ризиків повинні бути відображені в плані проекту. Визначення ризиків і розробка заходів щодо зменшення їхнього впливу на хід виконання проекту називається *керуванням ризиками.* Спрощено ризик можна розуміти як імовірність прояву яких-небудь несприятливих обставин, що негативно впливають на реалізацію проекту. Ризики можуть загрожувати проекту в цілому, створюваному програмному продукту або організації-розробнику. Можна виділити три типи ризиків.

1. *Ризики для проекту,* які впливають на графік робіт або ресурси, необхідні для виконання проекту.
2. *Ризики для розроблюваного продукту,* що впливають на якість або продуктивність розроблюваного програмного продукту.
3. *Бізнес-ризики,* що стосуються організації-розробника або постачальника.

Звичайно, ці типи ризиків можуть перетинатися. Наприклад, якщо досвідчений програміст залишає проект, це буде ризиком для проекту (оскільки затримується термін здачі готового продукту), ризиком для продукту (тому що новий програміст, що замінив попереднього, може виявитися не занадто досвідченим і зробити помилки в програмі) і бізнес-ризиком (оскільки затримка даного проекту може негативно вплинути на майбутні ділові контакти між замовником і організацією-розробником).

Конкретні типи ризиків, які можуть вплинути на даний проект, залежать від виду створюваного програмного продукту й від організаційного оточення, де реалізується програмний проект. Разом з тим багато типів ризиків здатні вплинути на будь-які програмні проекти, ці ризики наведені в табл.1.

Визначення ризиків. Визначаються можливі ризики для проекту, для розроблюваного продукту й бізнес-ризики.

Аналіз ризиків. Оцінюється ймовірність і послідовність появи ризикових ситуацій.

Планування ризиків. Плануються заходи щодо запобігання ризиків або мінімізації їхнього впливу на проект.

Моніторинг ризиків. Постійне оцінювання ймовірностей ризиків і виконання заходів щодо пом'якшення наслідків прояву ризикової ситуацій.

Таблиця 3.1 Можливі ризики програмних проектів

Ризик	Тип ризику	Опис ризику
Плинність розробників	Ризик для проекту	Досвідчені розробники залишають проект до його завершення
Зміна в керуванні організацією	Ризик для проекту	Організація змінює свої пріоритети в керуванні проектом
Неготовність апаратних засобів	Ризик для проекту	Апаратні засоби, які необхідні для проекту, не надійшли вчасно або не готові до експлуатації
Зміна вимог	Ризик для проекту й для розроблюваного продукту	Поява великої кількості непередбачених змін у вимогах, пропонованих до розроблюваного ПЗ
Затримка в розробці специфікації	Ризик для проекту й для розроблюваного продукту	Специфікації основних інтерфейсів підсистем не *надійшли* до розроблювачів відповідно до графіка робіт
Недооцінка розміру розроблюваної системи	Ризик для проекту й для розроблюваного продукту	Розмір системи значно перевищив первісну оцінку
Недостатня ефективність CASE-засобів	Ризик для проекту й для розроблюваного продукту	CASE-засоби, призначені для підтримки проекту, виявилися менш ефективними, ніж очікувалося
Зміни в технології розробки ПЗ	Ризик для розроблюваного продукту	Основні технології побудови програмної системи заміняються новими
Поява конкуруючого програмного продукту	Бізнес-ризик	На ринку програмних продуктів до закінчення проекту з'явилася конкуруюча програмна система

Процес керування ризиками, як і інші процеси планування, є ітераційним, виконуваним протягом усього строку реалізації проекту. Спочатку розробляються плани керування ризиками, потім постійно відслідковується ситуація навколо реалізації проекту. При надходженні нової інформації про можливі ризики заново проводиться аналіз ризиків і першорядна увага приділяється новим ризикам. У міру надходження нової інформації також змінюються плани заходів щодо запобігання й пом'якшення ризиків.

Результати процесу керування ризиками документуються у вигляді планів керування ризиками. Вони повинні включати опис можливих проектних ризиків, їхній аналіз і перелік заходів, необхідних для керування ризиками.

Визначення ризиків

Визначення ризиків – перша стадія процесу керування ризиками. На цій стадії описуються ризики, які можуть виявитися при реалізації проекту. У принципі на цій стадії не повинна оцінюватися ймовірність і значимість ризиків, але на практиці малоймовірні ризики з незначними наслідками зазвичай відкидаються відразу.

Визначення ризиків може виконуватися в режимі командної роботи з використанням підходу "мозковий штурм" або ґрунтуватися на досвіді менеджера. При визначенні ризиків може допомогти наведений нижче список можливих категорій ризиків.

1. *Технологічні ризики.* Виникають із програмних і апаратних технологій, на основі яких розробляється система.
2. *Ризики, пов'язані з персоналом.* Пов'язані зі членами команди розроблювачів.
3. *Організаційні ризики.* Виникають із організаційного оточення, у якому виконується проект.
4. *Інструментальні ризики.* Пов'язані з використовуваними CASE-засобами й іншими засобами підтримки процесу створення ПЗ.
5. *Ризики, пов'язані із системними вимогами.* Проявляються при зміні вимог, пред’явлених до розроблювальної системи.
6. *Ризики оцінювання.* Зв'язані оцінюванням характеристик програмної системи й ресурсів, необхідних для реалізації проекту.

У табл. 2 представлені деякі приклади, що ставляться до кожної з описаних категорій ризиків. Результатом етапу визначення ризиків буде довгий список можливих ризиків, які можуть вплинути на розроблюваний програмний продукт, проект або організацію-розробникавача.

Таблиця 3.2 Категорії ризиків

Категорія ризиків	Приклади ризиків
Технологічні	База даних, що використовується в програмній системі, не забезпечує обробку очікуваного обсягу транзакцій. Програмні компоненти, які використовуються повторно, мають дефекти, що обмежують їхні функціональні можливості
Пов'язані з персоналом	Неможливо підібрати працівників з необхідним професійним рівнем. Провідний розроблювач занедужав у самий критичний час. Неможливо організувати необхідне навчання персоналу
Організаційні	В організації, що виконує розробку ПЗ, відбулася реорганізація, у результаті чого змінилися пріоритети в керуванні проектом. Фінансові труднощі в організації привели до зменшення бюджету проекту
Інструмен-тальні	Програмний код, що генерується CASE-засобами, не ефективний. CASE-Засобу неможливо інтегрувати з іншими засобами підтримки проекту
Пов'язані із системними вимогами	Зміни вимог приводять до значних повторних робіт із проектування системи. Первісне нечітке формулювання користувальницьких вимог привело до значних змін системних вимог, що виявилися на пізніх стадіях розробки проекту

Аналіз ризиків

При аналізі для кожного певного ризику підраховується ймовірність його прояву й збиток, що він може нанести. Не існує простих методів виконання аналізу ризиків – значною мірою він заснований на думці й досвіді менеджера. Не претендуючи на виняткову точність, можна привести наступну шкалу ймовірностей ризиків і їхніх наслідків.

Імовірність ризику вважається дуже низкою, якщо вона має значення менш 10%; низкою, якщо її значення від 10 до 25 %; середньою при значеннях від 25 до 50%; високою, якщо значення коливається від 50 до 75%; дуже високою при значеннях більше 75%.

Можливий збиток від ризиків ситуацій можна підрозділити на катастрофічний, серйозний, терпимий і незначний.

Результати аналізу ризиків повинні бути представлені у вигляді таблиці ризиків, упорядкованих по ступені можливого збитку (табл.3).

Таблиця 3.3. Список ризиків після проведення їхнього аналізу

Ризик	Імовірність	Ступінь збитку
Фінансові труднощі в організації призвели до зменшення бюджету проекту	Низька	Катастро-фічний
Неможливо підібрати працівників із професійним рівнем, що вимагається	Висока	Катастро-фічний
Провідний розробник занедужав у найбільш критичний час	Середня	Серйозний
Програмні компоненти, використовувані повторно, мають дефекти, що обмежують їхні функціональні можливості	Середня	Серйозний
Зміни вимог призводять до значних повторних робіт із проектування системи	Середня	Серйозний
В організації, що виконує розробку ПЗ, відбулася реорганізація, у результаті чого змінилися пріоритети в керуванні проектом	Висока	Серйозний
База даних, що використовується в програмній системі, не забезпечує обробку очікуваного обсягу транзакцій	Середня	Серйозний
Недооцінки часу виконання проекту	Висока	Серйозний
CASE-засоби неможливо інтегрувати з іншими засобами підтримки проекту	Висока	Терпимий
Первісне нечітке формулювання користувальницьких вимог призвело до значних змін системних вимог, що виявилися на пізніх стадіях розробки проекту	Середня	Терпимий
Неможливо організувати необхідне навчання персоналу	Середня	Терпимий
Швидкість виявлення дефектів у системі нижча раніше запланованої	Середня	Терпимий
Розмір системи значно перевищує спочатку розрахований	Висока	Терпимий
Програмний код, згенерований CASE-засобами, неефективний	Середня	Незначний

Звичайно, як імовірність ризиків, так і можливий збиток від них повинні переглядатися при надходженні додаткової інформації про ці ризики й у міру

реалізації заходів щодо керування ними. Тому подібні таблиці ризиків повинні перероблятися на кожній ітерації процесу керування ризиками.

Після проведення аналізу ризиків визначаються найбільш значимі ризики, які потім відслідковуються протягом усього терміну виконання проекту. Визначення цих значимих ризиків залежить від їхніх імовірностей і можливого збитку. У загальному випадку завжди відслідковуються ризики з катастрофічними наслідками, а також ризики із серйозним збитком, значення ймовірності яких вище за середнє.

Вимоги до програмної системи часто класифікуються як функціональні, нефункціональні й вимоги предметної області.

1. *Функціональні вимоги.* Це перелік сервісів, які повинна виконувати система, причому повинне бути зазначене, як система реагує на ті або інші вхідні дані, як вона поводиться в певних ситуаціях і т.д. У деяких випадках вказується, що система не повинна робити.

2. *Нефункціональні вимоги.* Описують характеристики системи і її оточення, а не поводження системи. Тут також може бути наведений перелік обмежень, що накладаються на дії й функції, виконувані системою. Вони включають тимчасові обмеження, обмеження на процес розробки системи, стандарти й т.д.

3. *Вимоги предметної області.* Характеризують ту предметну область, де буде експлуатуватися система. Ці вимоги можуть бути функціональними й нефункціональними.

Функціональні вимоги

Ці вимоги описують поведінку системи й сервіси (функції), які вона виконує, і залежать від типу розроблювальної системи й від потреб користувачів. Якщо функціональні вимоги оформлені як користувальницькі, вони, як правило, описують системи в узагальненому виді. На противагу цьому функціональні вимоги, оформлені як системні, описують систему максимально докладно, включаючи її вхідного й вихідного дані, виключення й т.д.

Функціональні вимоги для програмних систем можуть бути описані різними способами. Розглянемо для прикладу функціональні вимоги до бібліотечної системи університету, призначеної для замовлення книг і документів з інших бібліотек.

1. Користувач повинен мати можливість проводити пошук необхідних йому книг і документів або по всій множині доступних каталожних баз даних або по певній їхній підмножині.

2. Система повинна надавати користувачеві належний засіб перегляду бібліотечних документів.

3. Кожне замовлення повинне бути наділений унікальним ідентифікатором (ORDER_ID), що копіюється у формуляр користувача для постійного зберігання.

Ці функціональні користувацькі вимоги визначають властивості, якими повинна володіти система. Вони взяті з документу, що містить користувацькі вимоги, і показують, що функціональні вимоги можуть бути описані з різним рівнем деталізації (порівняйте першу й третю вимоги).

Багато проблем, що виникають при розробці систем, пов'язані з неточністю й "розмитістю" специфікації вимог. Природно, розробники інтерпретують вимоги, що допускають двояке тлумачення, так, щоб систему було простіше реалізувати. Але це тлумачення може не збігатися з очікуваннями замовника. Така ситуація призводить до розробки нових вимог і внесення змін у систему. Це, у свою чергу, веде до затримки здачі готової системи і її подорожчанню.

Нефункціональні вимоги

Нефункціональні вимоги не пов’язані безпосередньо з функціями, виконуваними системою. Вони пов'язані з такими інтеграційними властивостями системи, як надійність, час відповіді або розмір системи. Крім того, нефункціональні вимоги можуть визначати обмеження на систему, наприклад на пропускну здатність пристроїв введення-виведення, або формати даних, використовуваних у системному інтерфейсі.

Багато нефункціональних вимог відносяться до системи в цілому, а не до окремих її засобів. Це означає, що вони більш значимі й критичні, ніж окремі функціональні вимоги. Помилка, допущена у функціональній вимозі, може знизити якість системи, помилка в нефункціональних вимогах може зробити систему непрацездатною.

Разом з тим, нефункціональні вимоги можуть ставитися не тільки до самої програмної системи: одні можуть ставитися до технологічного процесу створення ПЗ, інші – містити перелік стандартів якості, що накладаються на процес розробки. Крім того, у специфікації нефункціональних вимог може бути зазначено, що проектування системи повинне виконуватися тільки певними CASE-засобами, і наведено опис процесу проектування, якому необхідно слідувати.

Нефункціональні вимоги відображають користувальницькі потреби; при цьому вони ґрунтуються на бюджетних обмеженнях, ураховують організаційні можливості компанії-розроблювача й можливість взаємодії розроблювальної системи з іншими програмними й обчислювальними системами, а також такі зовнішні фактори, як правила техніки безпеки, законодавство про захист інтелектуальної власності й т.п. На рис.1 показана класифікація нефункціональних вимог.

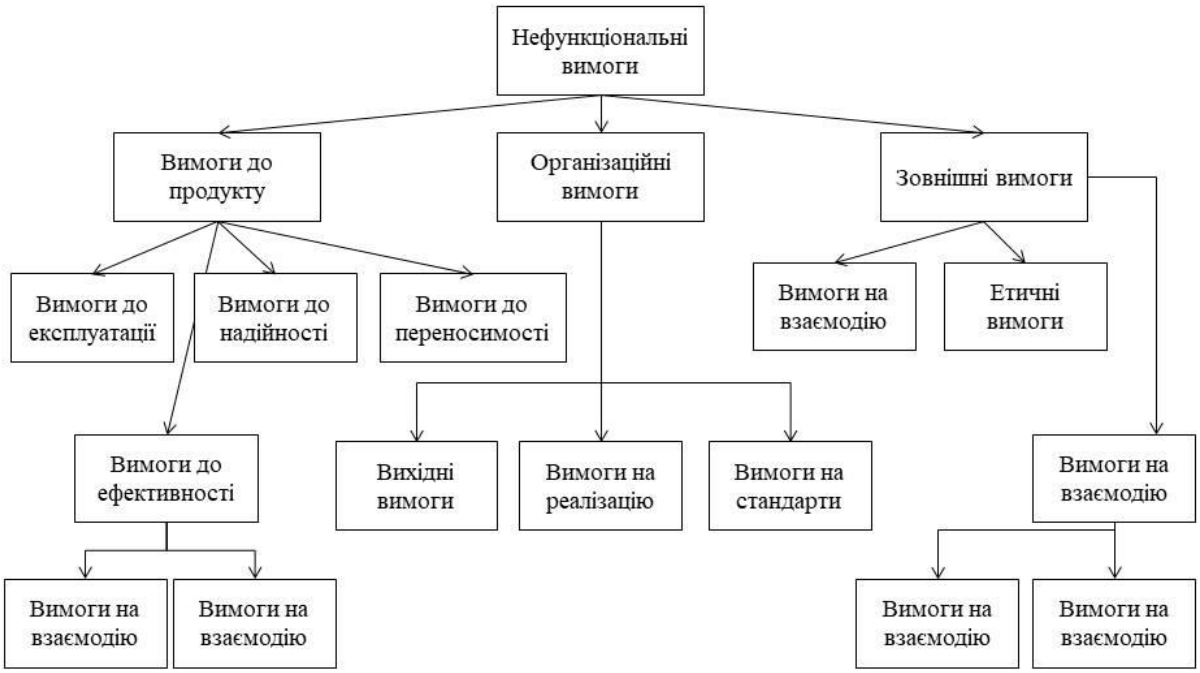

Рисунок 3.1 Типи нефункціональних вимог

Наведемо приклад вимог до продукту, організаційних і зовнішніх вимог. Вимоги до продукту позв'язані із середовищем програмування APSE для мови Ada. Це обмежує свободу проектувальника системи у виборі символів – можна використовувати тільки символи з користувальницького інтерфейсу APSE. Організаційні вимоги вказують, що система повинна розроблятися відповідно до внутрішнього стандарту компанії на розробку ПЗ, що має код XYZco-SP-STAN-95. Зовнішні вимоги випливають із необхідності дотримання законодавства про збереження конфіденційності. Внаслідок цього системні оператори не будуть мати доступ до тих даних, які їм не потрібні для роботи із системою.

Приклад нефункціональних вимог

Вимоги до продукту

Всі взаємодії між інтерфейсом APSE і користувачем здійснюються на основі стандартної безлічі символів мови Ada.

Організаційні вимоги

Розробка системи й створення супутньої документації виконуються на основі стандарту XYZCo-SP-STAN-95.

Зовнішні вимоги

Система не повинна розкривати конфіденційної інформації про замовника системи, крім його імені, а також телефонного номера системних операторів.

В ідеалі нефункціональні вимоги повинні виражатися через кількісні показники, які можна об'єктивно виміряти. У табл. 4 наведені показники, за

допомогою яких можна специфицифікувати нефункціональні системні властивості.

Таблиця 3.4. Кількісні показники для нефункціональних вимог

Показник	Одиниці виміру
Швидкість	Кількість виконаних транзакцій в секунду;
	час реакції на дії користувача;
	час відновлення екрана
Розмір	Кілобайти;
	кількість модулів пам'яті
Простота експлуатації	Час навчання персоналу;
	кількість статей у довідковій системі
Надійність	Середня тривалість часу між двома послідовними проявами помилок у системі;
	імовірність виходу системи з ладу;
	коефіцієнт готовності системи
Стійкість до збоїв	Час відновлення системи після збою;
	відсоток подій, що приводять до збоїв;
	імовірність псування даних при збоях
Переносимість	Відсоток машинно-залежних операторів;
	кількість машинно-залежних підсистем

Хід роботи

1. Опрацювати теоретичний матеріал до лабораторної роботи.
2. Визначити функціональні, не функціональні вимоги та ризики системи відповідно до власного варіанту (користуючись запропонованими прикладами та таблицями із теоретичної частини).
3. Сформулювати висновки.
4. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке ризик у процесі розробки інформаційної системи?
2. Які основні етапи процесу керування ризиками?
3. Що таке функціональні вимоги до інформаційної системи?
4. Що таке нефункціональні вимоги?
5. У чому полягає відмінність між функціональними та нефункціональними вимогами?
6. Яким чином можна мінімізувати ризики під час розробки програмного забезпечення?

Лабораторна робота №4. Вимоги предметної області. Вимоги користувача

Мета: Ознайомиться з вимогами предметної області. Навчиться розробляти вимоги предметної області. Ознайомиться з користувацькими вимогами. Навчиться розробляти користувацькі вимоги

Теоретичні відомості

Вимоги предметної області

Ці вимоги відображають умови, у яких буде експлуатуватися програмна система. Вони можуть бути представлені у вигляді нових функціональних вимог, у вигляді обмежень на вже сформульовані функціональні вимоги або у вигляді вказівок, як система повинна виконувати обчислення. Ці вимоги дуже важливі, оскільки відображають ту предметну область, де буде використовуватися дана система. Невиконання вимог предметної області може привести до виходу системи з ладу.

Як *приклад* розглянемо вимоги до бібліотечної системи.

1. Стандартний користувальницький інтерфейс, що надає доступ до всіх бібліотечних баз даних, повинен ґрунтуватися на стандарті Z39.50.

2. Для забезпечення авторських прав деякі документи повинні бути вилучені із системи відразу після одержання. Для цього, залежно від бажання користувача, ці документи можуть бути роздруковані або на локальному системному сервері, або на мережному принтері.

Перша вимога є обмеженням на системну функціональну вимогу. Вона вказує, що користувальницький інтерфейс до баз даних повинен бути реалізований відповідно до відповідного бібліотечного стандарту. Друга вимога є зовнішньою і спрямованою на виконання закону про авторські права, що застосовується до бібліотечних матеріалів. Із цієї вимоги випливає, що система повинна мати засіб "вилучити_на_друк", що застосовується автоматично для деяких типів бібліотечних документів.

Наведемо *приклад* вимоги предметної області, що вказує, як повинні виконуватися обчислення. Він взято зі специфікації системи автоматичного гальмування поїзда. Ця система повинна автоматично зупиняти поїзд на червоний сигнал семафора. Дана вимога вказує спосіб обчислення швидкості поїзда при гальмуванні. Тут використана термінологія, що застосовується при розрахунках швидкостей поїзда. Щоб розібратися в ній, необхідні відповідні знання про системи керування поїздами і їхніми характеристиками.

Приклад вимог предметної області

Гальмування поїзда обчислюється по формулі

Dпоїзд = Dкерування + Dградієнт,

де D градієнт дорівнює 9.81 м*з^2 * градієнт, що компенсує,/альфа. Значення 9.81 м*с^2/альфа відомо для всіх типів поїздів.

Наведений приклад показує основну проблему, пов'язану з вимогами предметної області. Вимоги цього типу використовують мову й позначення, властиві даній предметній області, що ускладнює їхнє розуміння розробниками ПЗ. Внаслідок цього, вимоги предметної області не завжди виконуються так, як мається на увазі замовниками програмної системи.

Користувацькі вимоги

Користувацькі вимоги до системи повинні описувати функціональні й нефункціональні системні вимоги так, щоб вони були зрозумілі навіть користувачеві, що не має спеціальних технічних знань. Ці вимоги повинні визначати тільки зовнішнє поводження системи, уникаючи по можливості визначення структурних характеристик системи. Користувацькі вимоги повинні бути написані природною мовою з використанням простих таблиць, а також наочних і зрозумілих діаграм.

Хід роботи

1. Опрацювати теоретичний матеріал до лабораторної роботи.
2. Визначити вимоги предметної області та вимоги користувача до системи, що відповідає вашому варіанту.
3. Сформулювати висновки.
4. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке предметна область інформаційної системи?
2. Які методи використовуються для збору вимог користувачів?
3. Що таке вимоги користувача та чим вони відрізняються від системних вимог?
4. Які етапи включає процес аналізу вимог?
5. Які помилки можуть виникати під час формування вимог?
6. Чому важливо правильно визначити вимоги предметної області?

Лабораторна робота №5. Побудова опорних точок зору на основі методу VORD для формування аналізу вимог. Складання сценаріїв основних подій

Мета: Навчитись застосовувати метод VORD при проектування програмного забезпечення. Навчитись складати сценарії подій.

Теоретичні відомості

Теоретичні відомості детально описані у конспекті лекцій з даної теми.

Вимоги до результатів виконання лабораторної роботи:

- наявність діаграми ідентифікації точок зору і діаграми ієрархії точок зору;
- наявність сценаріїв подій (послідовність дій).

Хід роботи

1. Вивчити теоретичний матеріал по темі (конспект лекцій).
2. Побудувати опорні точки зору на основі методу VORD для формування і аналізу вимог. Результатом повинні бути дві діаграми: діаграма ідентифікації точок зору та діаграма ієрархії точок зору.
3. Скласти сценарії основних подій.
4. Скласти інформаційну модель майбутньої системи, що включає в себе опис основних об'єктів системи і взаємодію між ними. На основі отриманої інформаційної моделі і діаграм ідентифікації точок зору, діаграми ієрархії точок зору.
5. Сформулювати висновки щодо ефективності застосованого методу.
6. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке метод VORD і для чого він використовується?
2. Що таке точка зору (viewpoint) у методі VORD?
3. Які етапи включає метод VORD?
4. Що таке сценарій подій у процесі аналізу вимог?
5. Яким чином сценарії допомагають описувати поведінку системи?
6. Які переваги використання методу VORD під час аналізу вимог?

Лабораторна робота №6. Розробка технічного завдання

Мета: Засвоїти основні принципи та засади оформлення технічного завдання на розробку програмного забезпечення.

Теоретичні відомості

Призначення технічного завдання

Технічне завдання (ТЗ) є основним документом усього проекту та усіх взаємовідносин замовника і розробника. Коректне ТЗ, написане й погоджене усіма зацікавленими та відповідальними особами, є запорукою успішної реалізації проекту.

Великі проекти вимагають серйозного проектного дослідження. Як правило, на ці дослідження виділяється окремий бюджет і часом не менший, ніж на безпосередньо розробку проекту. Часто доводиться проектні дослідження зводити до мінімуму або частину досліджень проводити безкоштовно в надії на одержання великого проекту, що в остаточному підсумку може негативно позначитися на ефективності робіт.

Як правило, замовник не є професіоналом в області високих технологій, і завдання ним ставиться в загальному вигляді. У такому випадку виконавець може сам запропонувати варіанти, черговість і етапність розв’язання поставленого завдання.

Структура технічного завдання

Після того як обговорено основні завдання й загалом стає зрозуміло, чого прагне замовник, необхідно скласти й погодити технічне завдання.

Згідно з ГОСТом 19.201-78 ТЗ повинно містити такі розділи:

- вступ;
- підстави для розробки;
- призначення розробки;
- вимоги до програми чи програмному виробу;
- вимоги до програмної документації;
- техніко-економічні показники;
- стадії та етапи розробки;
- порядок контролю та приймання.

У ТЗ допускається включати додатки.

У залежності від особливостей програми чи програмного виробу допускається уточнювати зміст розділів, вводити нові розділи чи поєднувати окремі з них.

У розділі «Вступ» вказують найменування, коротку характеристику області застосування програми чи програмного виробу та об’єкта, у якому використовують програму чи програмний виріб.

У розділі «Підстави для розробки» повинно бути зазначено:

– документ (документи), на підставі якого ведеться розробка;
– організація, що затвердила цей документ, і дата його затвердження;
– найменування і (або) умовне позначення теми розробки.

У розділі «Призначення розробки» повинно бути зазначене функціональне та експлуатаційне призначення програми чи програмного виробу.

Розділ «Вимоги до програми чи програмного виробу» повинен містити наступні підрозділи:

– вимоги до функціональних характеристик;
– вимоги до надійності;
– умови експлуатації;
– вимоги до складу та параметрів технічних засобів;
– вимоги до інформаційної та програмної сумісності;
– вимоги до маркування та упакування;
– вимоги до транспортування та збереження;
– спеціальні вимоги.

У підрозділі «Вимоги до функціональних характеристик» повинно бути зазначено вимоги до складу виконуваних функцій, організації вхідних і вихідних даних (перелічені всі вхідні й вихідні дані та зазначено всі відомі вимоги до їх організації), тимчасових характеристик тощо. У даному підрозділі обов’язково має бути зазначено, якими групами користувачів та яким чином (з зазначенням відповідних функцій програми) планується використання програмного забезпечення.

У підрозділі «Вимоги до надійності» повинно бути зазначено вимоги до забезпечення надійного функціонування (забезпечення стійкого функціонування, контролю початкової та вихідної інформації, часу відновлення після відмови тощо). У даному підрозділі визначається, які саме збої, ситуації невірного введення даних користувачем тощо мають оброблятися програмою.

У підрозділі «Умови експлуатації» повинно бути зазначено умови експлуатації (температура навколишнього повітря, відносна вологість тощо для обраних типів носіїв даних), при яких повинні забезпечуватися задані характеристики, вид обслуговування, необхідна кількість і кваліфікація персоналу (з зазначенням задач, які такий персонал повинен виконувати).

У підрозділі «Вимоги до складу і параметрів технічних засобів» указують необхідний склад технічних засобів із зазначенням їхніх основних технічних характеристик.

У підрозділі «Вимоги до інформаційної і програмної сумісності» повинно бути зазначено вимоги до інформаційних структур на вході і виході та методів розв’язання, вихідних кодів, мов програмування та програмних засобів, що

використовуються програмою. За необхідності повинен забезпечуватися захист інформації та програм.

У підрозділі «Вимоги до маркування та упакування» у загальному випадку указують вимоги до маркування програмного виробу, варіанти та способи упакування.

У підрозділі «Вимоги до транспортування та збереження» повинні бути зазначені для програмного виробу умови транспортування, місця збереження, умови збереження, умови складування, терміни збереження в різних умовах.

У підрозділі «Спеціальні вимоги» у даній лабораторній роботі потрібно навести вимоги до графічного інтерфейсу користувача. Для цього потрібно виконати попереднє проектування інтерфейсу. Для таких цілей існує багато доволі простих і зручних програмних засобів, частина з яких є безкоштовними. Наприклад, програма Pencil надає набір інструментів для проектування інтерфейсів десктопних систем, мобільних додатків, веб-додатків.

У розділі «Вимоги до програмної документації» повинен бути зазначений попередній склад програмної документації і, за необхідності, спеціальні вимоги до неї.

У розділі «Техніко-економічні показники» повинні бути зазначені: орієнтована економічна ефективність, передбачувана річна потреба, економічні переваги розробки порівняно з кращими вітчизняними та закордонними аналогами.

У розділі «Стадії та етапи розробки» установлюють необхідні стадії розроблення, етапи та зміст робіт (перелік програмних документів, що повинні бути розроблені, погоджені та затверджені), а також, як правило, терміни розробки, визначають виконавців.

У розділі «Порядок контролю та приймання» повинні бути зазначені види тестів і загальні вимоги до приймання роботи. У даному розділі можуть такою зазначатися конкретні випробування, які мають проводитися для контролю та приймання програмного продукту.

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями, необхідними для написання ТЗ на розробку програмного забезпечення.

2. Узгодити з викладачем індивідуальне завдання для розроблення програмного забезпечення. Необхідно враховувати, що отримана тема та відповідні завдання мають ґрунтуватися на наступних принципах: розроблювана у подальшому система має працювати з певним сховищем даних, має виконувати обробку та збереження даних, повинна забезпечуватися можливість користування великою кількістю різних груп користувачів, а також повинна мати візуальний користувацький інтерфейс.

3. Розробити ТЗ у відповідності до отриманого індивідуального завдання, а також відповідно до всіх вимог, які висуваються до ТЗ, та викладених теоретичних відомостей.

4. Провести критичний аналіз розробленого ТЗ на предмет відображення всіх вимог до розроблюваного програмного забезпечення та єдиного підходу до їх формулювання, а також на предмет правильно встановлених часових меж розробки. За необхідності внести корективи.

5. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке технічне завдання на розробку інформаційної системи?
2. Які основні розділи містить технічне завдання?
3. Яку роль відіграє технічне завдання у процесі розробки програмного забезпечення?
4. Які вимоги повинні бути відображені у технічному завданні?
5. Хто бере участь у розробці та погодженні технічного завдання?
6. Чому важливо чітко формулювати вимоги у технічному завданні?

Лабораторна робота №7. Побудова діаграм варіантів використання (Usecase Diagrams)

Мета: Одержати практичні навики *у* побудові діаграм варіантів використання для обраного варіанту комп'ютерної системи.

Теоретичні відомості

Діаграми варіантів використання (Usecase Diagrams) використовуються для відображення сценаріїв використання системи (Usecase) та користувачів системи (actors), які використовують її функції.

Актори на діаграмі варіантів використання позначаються символом людини, а варіанти використання – еліпсом. Актори та варіанти використання поєднуються асоціацією. Також актори можуть поєднуватися з використанням зв'язків узагальнення. На рис. 1 показано фрагмент діаграми варіантів використання для інтернет-магазину. Актор «Покупець» при цьому може виконати сценарій «Замовити товар».

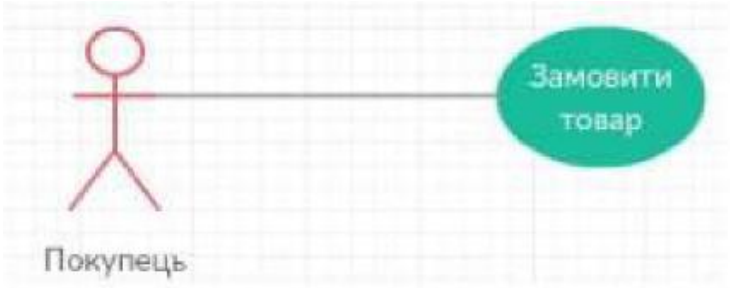

Рисунок 7.1 – Фрагмент діаграми варіантів використання

Варіанти використання можуть бути пов'язані між собою трьома видами зв'язків: узагальненням (generalization), розширенням (extend relationship) та включенням (include relationship).

Відношення узагальнення (generalization) показують відношення між загальним і частковим. Наприклад, на рис. 2 варіанти використання «Знайти за категорією» та «Знайти за виробником» є частковими випадками загального варіанта «Знайти товар», тому вони поєднані даним відношенням. Також дане відношення може використовуватися для поєднання акторів. Актор «Адміністратор» може виконувати всі функції актора «Покупець», тобто виступає частковим випадком покупця, але може виконувати і специфічні операції (варіант використання «Перевірити інформацію бази даних»).

Відношення включення (include) відображає зв'язок «ціле — частина», тобто один варіант завжди в певний момент виконання повністю включає інший. Для прикладу частиною варіанта використання «Замовити товар» є сценарії «Знайти товар» та «Вибрати спосіб оплати», оскільки для того, щоб замовити товар, покупець завжди має відшукати його в каталозі та обрати метод оплати.

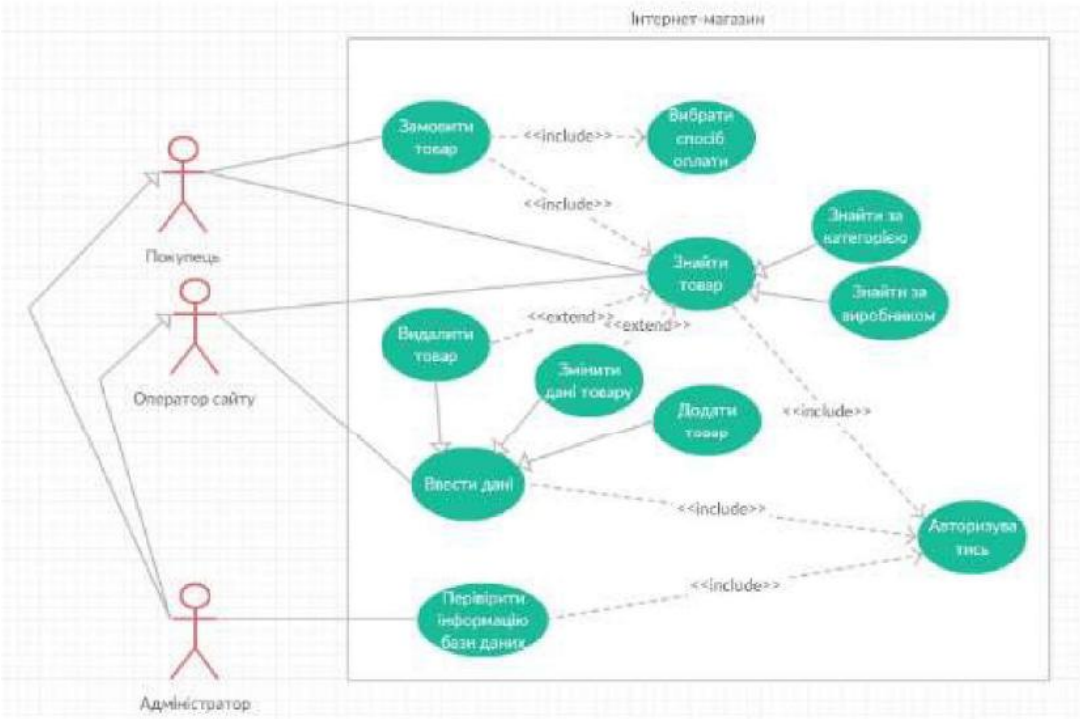

Рисунок 7.2 – Повна діаграма варіантів використання для Інтернет-магазину

Відношення розширення (extend) визначає такий тин відношення, коли один варіант за певних умов повністю використовує інший (розширює його). Так, наприклад, оператор Інтернет-магазину може видалити товар («Видалити товар»), знаючи його ідентифікатор, або провівши попередньо пошук товару («Знайти товар»).

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями.
2. З використанням обраного веб-сервісу (наприклад, creately.com, draw.io) або програмного продукту (BPwin, Rational Rose) створити діаграму варіантів використання для власного варіанту комп'ютерної системи. Діаграма повинна містити усіх акторів (користувачів системи) та по три варіанти використання для кожного актора. Пов’язати варіанти використання та акторів, при цьому використати усі види зв'язків (association, generalization, extend relationship, include relationship).
3. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.

4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке діаграма варіантів використання?
2. Які основні елементи діаграми Use Case?
3. Хто такі актори у діаграмах варіантів використання?
4. Які типи зв'язків використовуються у Use Case діаграмах?
5. Яку роль відіграють діаграми варіантів використання у процесі проєктування системи?
6. Які правила побудови Use Case діаграм?

Лабораторна робота №8. Побудова діаграм взаємодії (Interaction Diagrams)

Мета: Одержати практичні навики у побудові діаграм взаємодії для обраного варіанту комп'ютерної системи.

Теоретичні відомості

Діаграми взаємодії (Interaction Diagrams) відображають взаємодію логічних елементів системи між собою у процесі виконання актором певного варіанта використання. Розрізняють два типи діаграм взаємодії: діаграми послідовності (Sequence Diagrams) та кооперації (Collaboration Diagrams), які є різними представленнями одного і того ж процесу.

Головними елементами діаграм послідовності є об’єкти, які є логічними сутностями, що представляють окремі елементи системи та повідомлення, якими вони обмінюються. В якості об’єктів можуть виступати також актори. Повідомлення можуть бути не тільки абстрактними діями, що виконуються, але і методи класів, створених на діаграмі класів. Повідомлення на діаграмі послідовності пронумеровані, тобто мають чітку послідовність.

Для прикладу розглянемо систему електронного документообігу на підприємстві. Для побудови діаграми послідовності оберемо варіант використання «Створити документ», який виконує актор «Користувач» (рис. 8.1).

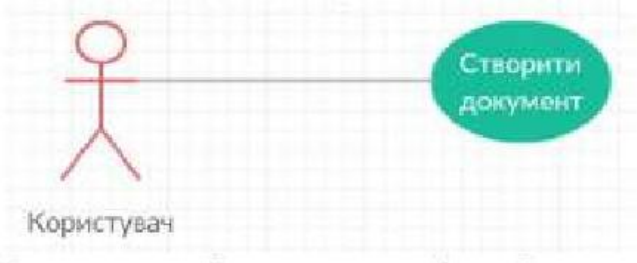

Рисунок 8.1 – Фрагмент діаграми варіантів використання для системи документообігу

Наведемо опис головного успішного сценарію даного варіанта використання у короткій формі.

Користувач має на екрані форму із запитом на створення документа, він обирає шаблон документа зі списку шаблонів, задає шлях для збереження та назву документа; система зберігає введені дані в якості параметрів документа, запускає прикладне програмне забезпечення, яке відповідає формату документа (наприклад MS Word, MS Excel і т. д,) та створює документ із відповідною назвою.

На рис. 2 наведено діаграму послідовності для даного прикладу. У верхній частині діаграми наведено перелік об’єктів (логічні сутності), які взаємодіють між собою у процесі виконання сценарію. Часова шкала на даній діаграмі направлена згори донизу, крім того, повідомлення пронумеровані відповідно до черги їх

пересилання між об’єктами. Нижче наведено короткий опис подій, що відбуваються при виконанні даного варіанта використання.

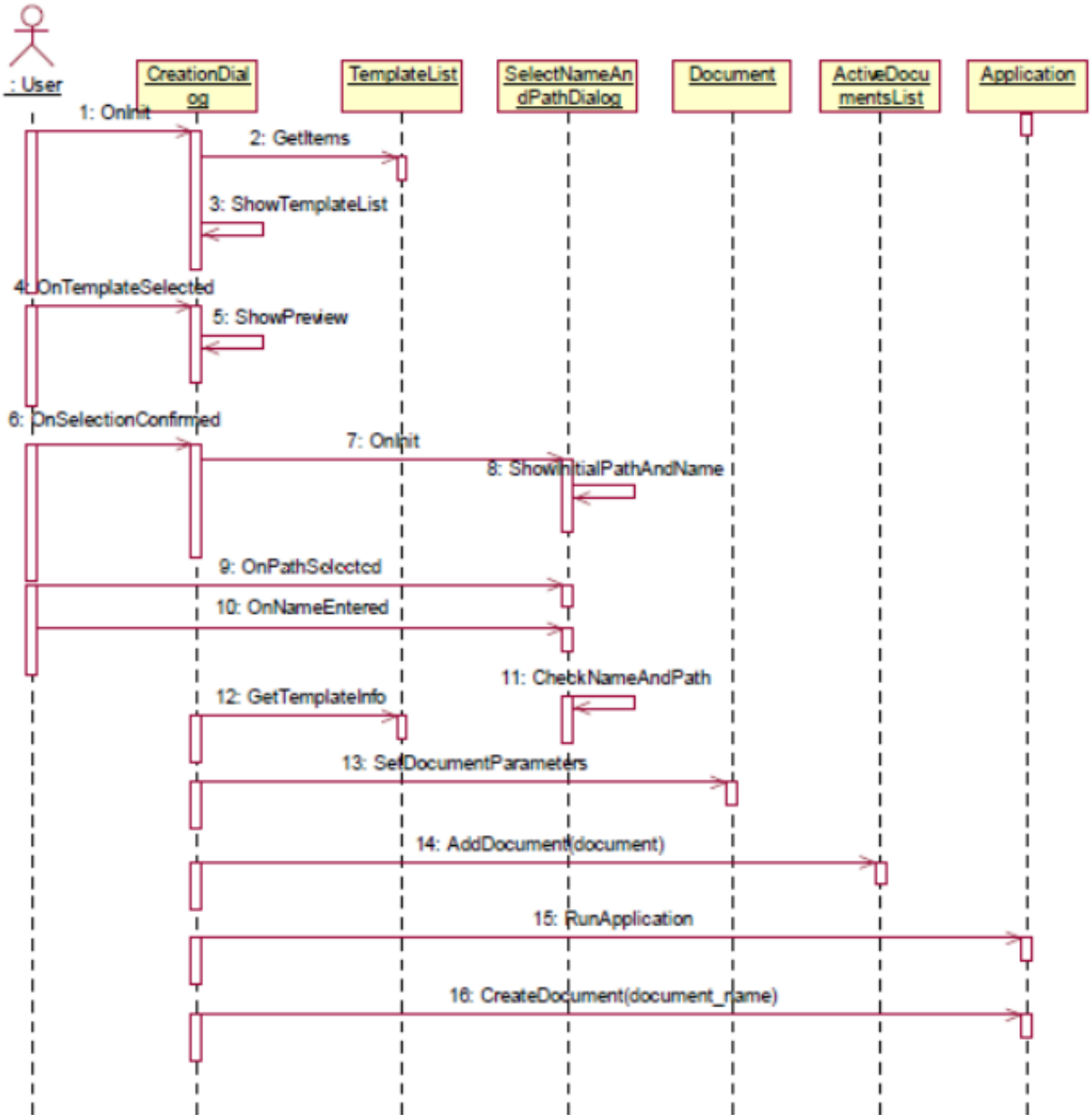

Рисунок 8.2 – Діаграма послідовності для варіанта використання «Створити новий документ»

При ініціалізації діалогу для створення нового документа (1) завантажується (2) та відображається (3) список шаблонів документів. При виборі користувачем одного з шаблонів (4), в діалозі відображається його початковий вигляд (5). Після того, як користувач підтверджує свій вибір остаточно, наприклад, скориставшись методом DoubleClick (6), система ініціалізує діалог (7), де просить ввести шлях до директорії та ім’я файла, у якому документ буде зберігатися. При цьому відображаються default значення для шляху та імені файла

(8), Користувач вводить шлях та ім'я файла (9), система перевіряє введені дані (10). Система зчитує інформацію шаблона (11) і записує всі параметри до об’єкта, який представляє документ (12). Потім документ додається до списку активних документів (13) і відповідно до параметрів документа запускається прикладна програма, яка відповідає формату шаблона документа (14). При цьому в прикладній програмі створюється новий документ з іменем, яке ввів користувач (15) (рис. 2).

Додатковими елементами діаграми послідовності є лінії життя об’єктів (довгі вертикальні пунктирні лінії), які відображають час життя об’єкта від його створення до знищення; фокус керування (прямокутники на лініях життя об’єктів) відображає, який об'єкт виконує операції в певний момент часу; та символи знищення об’єктів (хрест на лінії життя об’єкта), що відображає процес знищення об’єкта (рис. 3). Наприклад, об’єкт «Authorization Record» на діаграмі створюється на 2-му кроці виконання сценарію, використовується протягом 4-6 кроків та знищується після шостого кроку.

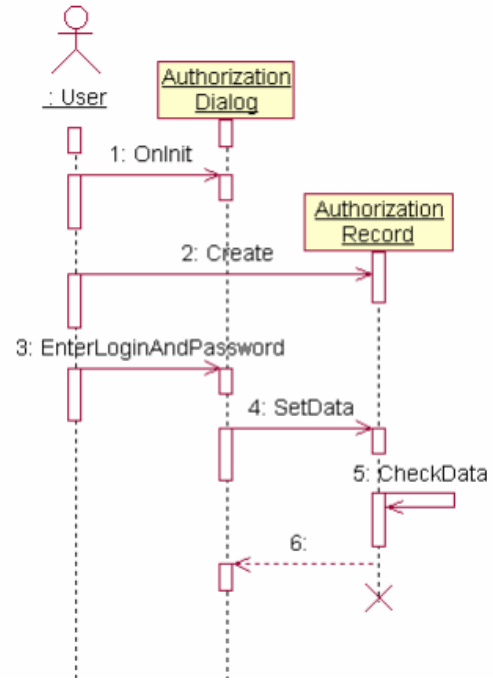

Рисунок 8.3 – Фрагмент діаграми послідовності для процесу авторизації користувача

Основні типи повідомлень на діаграмі послідовності:

– пряме – повідомлення, яке об’єкт-ініціатор надсилає об'єкту-приймачу (1-4);

– рефлексивне, яке об’єкт надсилає сам собі (5);

– зворотнє, при якому управління повертається об’єкту ініціатору (6), часто повідомлення даного типу не мають назви.

Діаграма кооперації має те ж саме призначення, що і діаграма послідовності, але відображає процес виконання варіанта використання в іншій нотації (рис. 4). Її головними компонентами є об’єкти, назви повідомлень та їх напрям. Характеристика даних компонентів була надана у процесі опису діаграми послідовності.

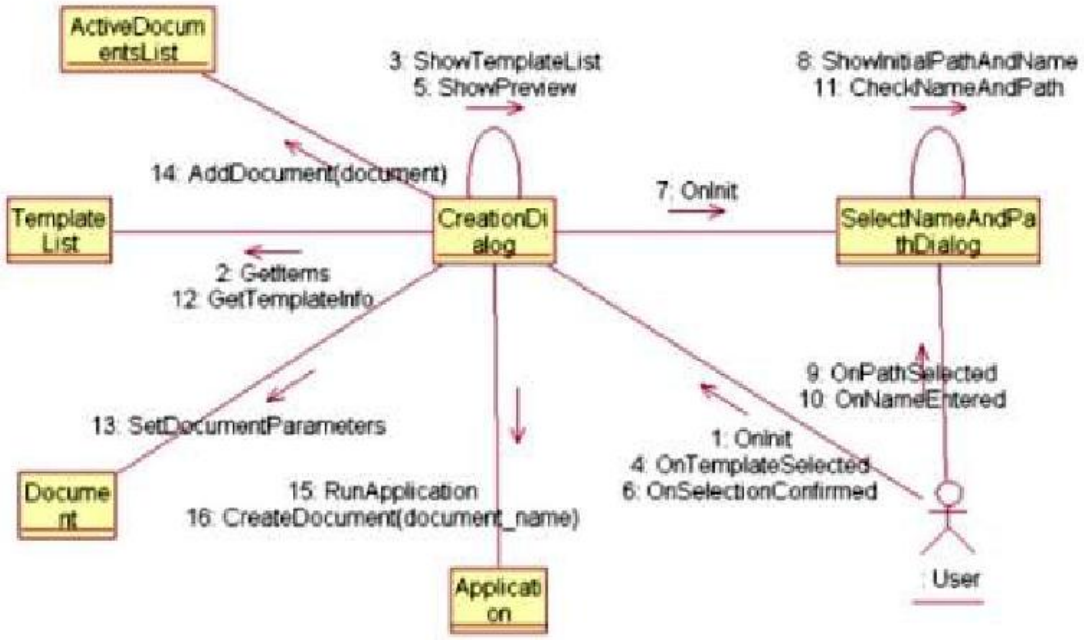

Рисунок 8.4 – Діаграма кооперації для варіанта використання «Створити новий документ»

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями.
2. Для кожного варіанта використання на Usecase Diagram створити Sequence або Collaboration Diagram (тобто у проекті повинно бути не менше шести діаграм кооперації та послідовності). На кожній діаграмі взаємодії повинен бути головний актор (при наявності) та не менше 5 об’єктів. Кожна діаграма взаємодії повинна містити не менше 10 повідомлень, якими обмінюються об'єкти в процесі виконання сценарію. Загальна сума різних об’єктів у проекті повинна налічувати 12-15 об’єктів. Об’єкти та повідомлення на діаграмах повинні мати зрозумілі назви. При побудові діаграм використовувати прямі, рефлексивні та зворотні типи повідомлень, а також символи знищення об’єктів.
3. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.

3. Покроковий опис виконання завдання.

4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке діаграми взаємодії в UML?
2. Які типи діаграм взаємодії існують?
3. Що відображає діаграма послідовності (Sequence Diagram)?
4. Які основні елементи використовуються у діаграмах взаємодії?
5. Яке призначення повідомлень у діаграмах взаємодії?
6. Яку роль відіграють діаграми взаємодії у моделюванні поведінки системи?

Лабораторна робота №9. Діаграми станів та переходів

Мета: Одержати практичні навики у побудові діаграм станів та переходів для обраного варіанту комп'ютерної системи.

Теоретичні відомості

Діаграми станів та переходів (Statechart Diagrams) разом із діаграмами діяльності та взаємодії, відображають певний сценарій, що виконується у процесі функціонування системи в цілому, або певної її частини.

Діаграма станів відображає скінчений автомат у вигляді графу, вершинами якого є стани об'єкта, поведінка якого моделюється, а переходами – події, які переводять об'єкт, який розглядається, з одного стану в інший. При цьому вважається, що час перебування об'єкта в певному стані набагато більший за час, необхідний для переходу з одного стану в інший, тобто переходи між станами здійснюються миттєво.

Стан (state) – це логічна сутність, що використовується для моделювання певної ситуації, дії, процесу. Кожен стан має ім'я та список внутрішніх дій. В якості імені стану найчастіше використовуються іменники, наприклад: «Введення паролю», «Очікування», «Перевірка параметрів». Список внутрішніх дій містить перелік дій, які виконуються у процесі знаходження системи чи об'єкта в даному стані. Кожна дія відображається у форматі:

<період виконання>/<назва дії>,

де поле <період виконання> може набувати наступних значень:

OnEntry – дія виконується під час того, як система входить у даний стан;

OnExit – дія виконується при виході з даного стану;

Do – дія виконується під час знаходження в даному стані;

OnEvent – дія виконується при настанні певної (зовнішньої) події.

Графічне представлення стану «Введення паролю» з трьома внутрішніми діями наведено на рис. 1, Перехід у даний стан ініціюється при введенні користувачем символів логіну. Для кожного введеного символу система зчитує рядок логіну, додає додатковий символ до неї і зберігає її.

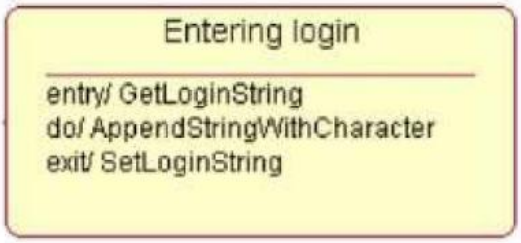

Рисунок 9.1 – Графічне представлення стану

Представлення початкового та кіцевого станів на Statechart Diagrams показано на рис. 9.2. Існує можливість створити також один із специфічних станів:

1 Вхідний стан – стан, в якому знаходиться система (об’єкт) у початковий момент часу.

2 Вихідний стан – стан, в якому знаходиться система (об'єкт) в момент закінчення виконання певної послідовності дій.

3 Стан історії – стан, який запам’ятовує дані, що використовувалися при попередньому входженні системи в даний стан.

Рисунок 9.2 – Представлення початкового та кінцевого станів на Statechart Diagrams

Рис. 9.3 демонструє стан історії «Перевірка параметрів дзвінка». Після перевірки параметрів перед здійсненням виклику запам’ятовується номер абонента для можливості його повторного виклику в майбутньому.

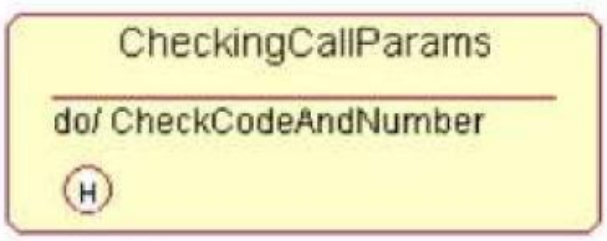

Рисунок 9.3 – Графічне представлення стану історії

Переходи (transitions) на Statechart Diagrams представлені стрілкою, що виходить з попереднього стану і входить у наступний. Кожен перехід має наступну специфікацію:

<тригер>(<параметри>) [<гранична умова>]/<дія>,

де <тригер> – подія, що ініціює можливість переходу;

(<параметри>) – параметри події;

[<гранична умова>] – умова, необхідна для здійснення переходу;

<дія> - дія, що виконується у процесі переходу.

На Statechart Diagrams можна задати два типи переходів:

1. Звичайний – перехід з одного стану в інший.
2. Рефлексивний – перехід із даного стану в цей же стан (зображається у вигляді петлі на графі).

На рис. 9.4 представлений фрагмент діаграми станів і переходів із застосуванням різних типів переходів для системи «Міні-АТС». Перехід із початкового стану у стан «Очікування» відбувається при ввімкненні системи, за умови її успішної ініціалізації. При цьому виконується дія Tone (подання тонового сигналу). Рефлексивний перехід для стану введення коду міста виникає при введенні нової цифри, параметр int d символізує код цифри.

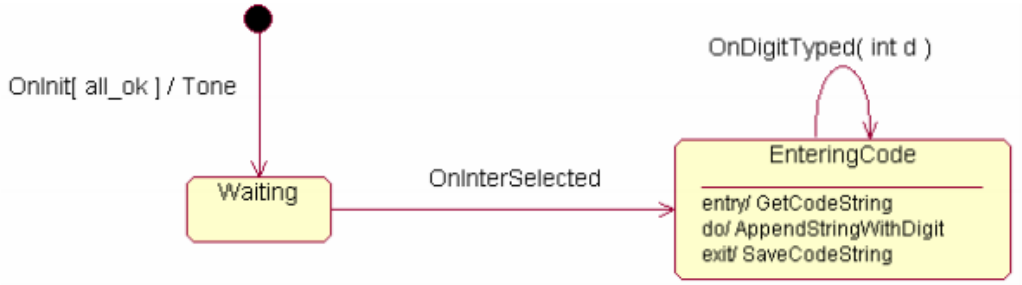

Рисунок 9.4 Ф- рагмент діаграми станів та переходів для Міні-АТС

На рис. 9.5 зображено приклад діаграми станів та переходів для конкретного об’єкта (діалогу авторизації певної системи). Рис. 9.6 представляє діаграму станів та переходів для Міні-АТС.

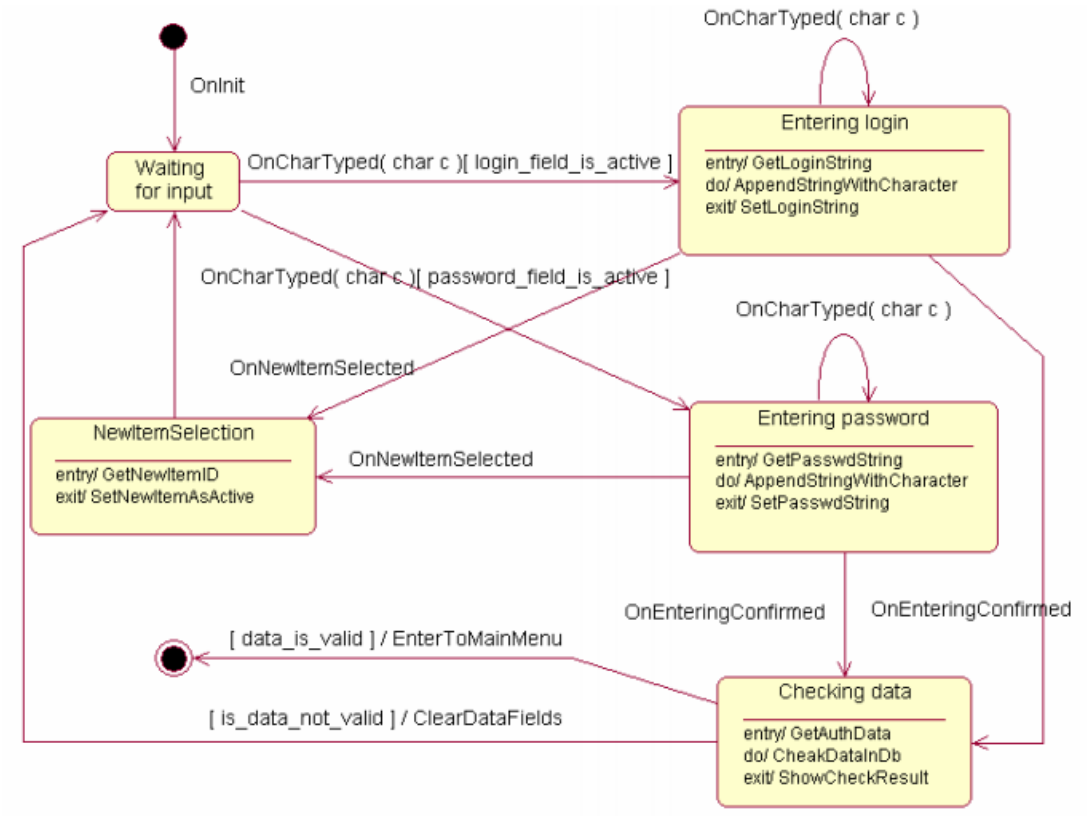

Рисунок 9.5 – Приклад діаграми станів та переходів для авторизації користувача в системі

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями.
2. Створити одну діаграму станів для опису процесу функціонування обраної системи в цілому і дві діаграми для конкретних елементів системи. Використовувати діаграму станів для авторизації користувачів забороняється.

Вимоги:

1) Кожна діаграма повинна містити не менше 6 станів.

2) По можливості використати обидва типи переходів (звичайний і рефлексивний).

3) Для кожного переходу визначити хоча б одну з характеристик (тригер, гранична умова, дія).

3. Оформити звіт з лабораторної роботи.

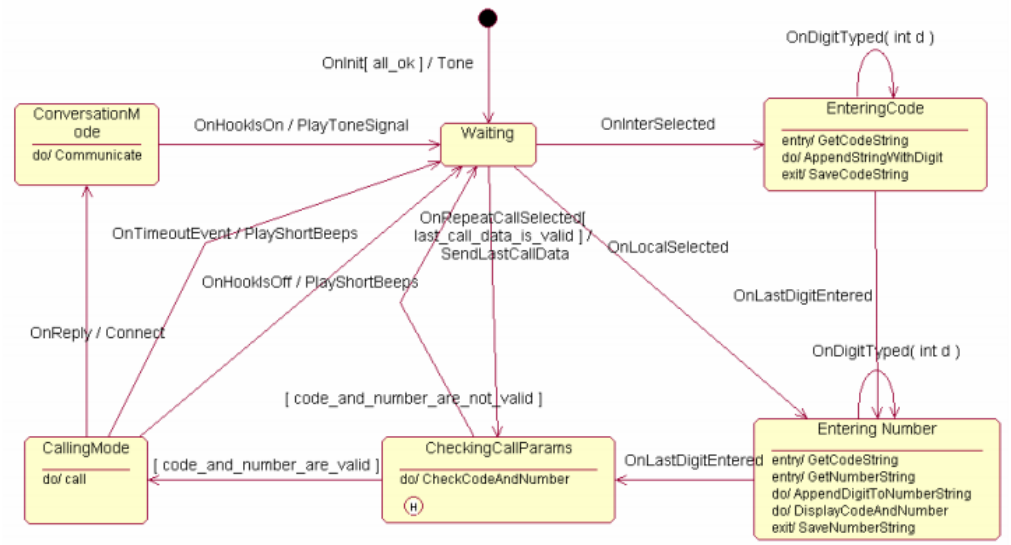

Рисунок 9.6 – Приклад діаграми станів та переходів для системи «Міні-АТС»

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке діаграма станів?
2. Які основні елементи містить діаграма станів?
3. Що таке стан об'єкта?
4. Що таке перехід між станами?
5. У яких випадках доцільно використовувати діаграми станів?
6. Яку інформацію можна отримати з діаграми станів системи?

Лабораторна робота №10. Побудова діаграм діяльності

Мета: Одержати практичні навики у побудові діаграм діяльності для обраного варіанту комп'ютерної системи.

Теоретичні відомості

Діаграми діяльності (activity diagrams) відображають послідовність дій, що виконується в процесі реалізації певного варіанта використання або функціонування системи в цілому. Діаграми діяльності є аналогом блок-схеми будь-якого алгоритму. Вони, як і діаграми станів та переходів, відображаються у вигляді орієнтованого графу, вершинами якого є дії, а ребрами – переходи між діями.

Діяльність (activity) є частковим випадком стану (state) без назви, який має одну вхідну подію (OnEntry action). Тому для кожної діяльності назва складається з дієслова та декількох пояснюючих слів, наприклад «Розрахувати заробітну платню» чи «Перевірити результати запиту». Графічне зображення діяльності «Перекласти слово» подано на рис. 1.

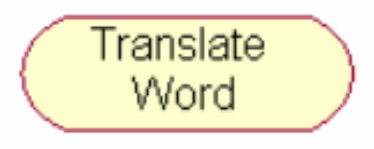

Рисунок 10.1 – Графічне представлення елементу Activity на діаграмі діяльності

Події (events) на переходах діаграми діяльності не задаються, оскільки вважається, що перехід від однієї дії до іншої здійснюється безумовно. Гранична умова (guard condition) використовується лише для визначення дії, до якої переходить керування у випадку неоднозначності (рис.10.2). Тобто, якщо з даної вершини на діаграмі діяльності можна перейти до декількох інших вершин для всіх переходів необхідно визначити граничну умову. Характеристика дії (action) для переходу також не має сенсу, оскільки всі дії на цій діаграмі представлені вершинами графу.

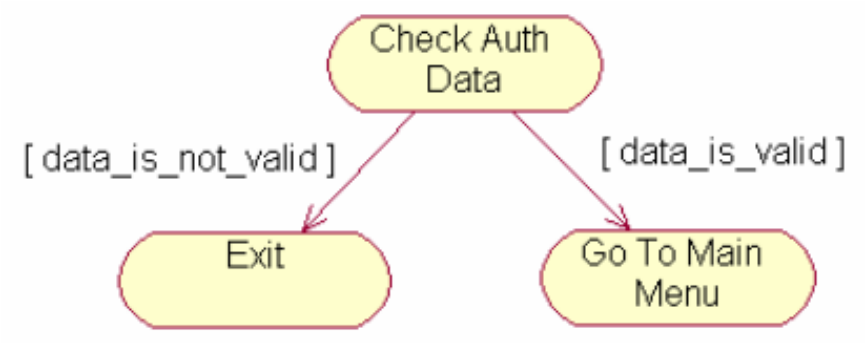

Рисунок 10.2 – Фрагмент діаграми діяльності для процесу авторизації

Для діаграми діяльності характерними є наступні спеціальні стани:

1 Початковий стан – аналогічний до діаграми станів та переходів.

2 Кінцевий стан – аналогічний до діаграми станів та переходів.

3 Стан прийняття рішення – стан, в якому здійснюється прийняття рішення про пер єн вправлення потоку управління до одного зі станів, пов’язаних із даним станом.

4 Стан синхронізації – стан, в якому здійснюється розділення загального потоку управління на декілька гілок (чи навпаки, декілька гілок поєднуються в єдиний потік).

Спеціальні стани прийняття рішення та синхронізації представлені на рис. 10.3 а та 3 б відповідно.

Рисунок 10.3 Графічне представлення спеціальних станів діаграми діяльності

Розглянемо застосування спеціальних станів на конкретному прикладі (варіант використання «Перекласти слово» для електронного словника).

На рис. 10.4 показано повний вигляд діаграми діяльності.

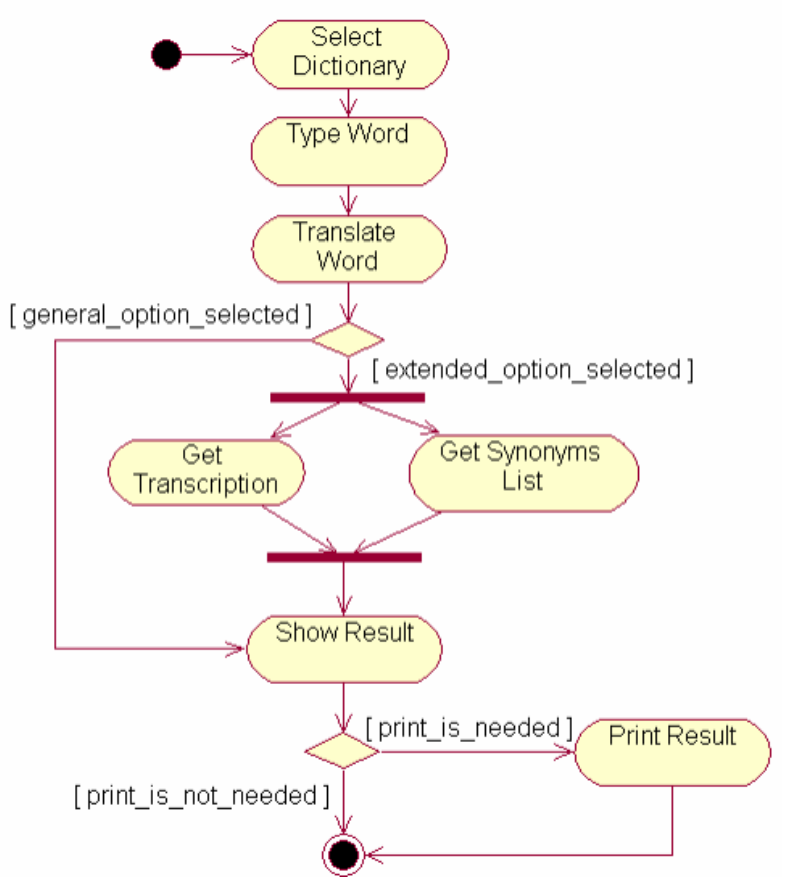

Рисунок 10.4 – Діаграма діяльності для перекладу слова електронним словником

На початковому етапі перекладу обирається словник, користувач вводить слово та система робить його переклад, далі з використанням стану прийняття рішення визначається, чи обрані додаткові опції перекладу. У випадку, якщо опцій не обрано, система переходить до показу результату перекладу. В іншому випадку, потік керування розподіляється на дві гілки, кожна з яких виконує певну дію (отримати транскрипцію слова та список синонімів відповідно). Після закінчення виконання обох операцій дві гілки поєднуються в єдиний потік і здійснюється показ результату. Потім визначається, чи потрібен друк для отриманої інформації, і у випадку необхідності вона друкується.

Також діаграми діяльності використовуються для відображення послідовності дій при моделюванні бізнес-процесів. При цьому використовується додатковий елемент діаграми, який має назву swimlane. Swimlane в дослівному перекладі означає «доріжка плавального басейну» (за аналогією з графічним відображенням). В якості swimlanes на діаграмі можуть виступати фізичні особи, групи осіб, відділи підприємства, чи навіть окремі організації. Розглянемо діаграму діяльності зі swimlanes для спрощеного варіанта бізнес-процесу «Розробка програмного забезпечення» (рис. 10.5).

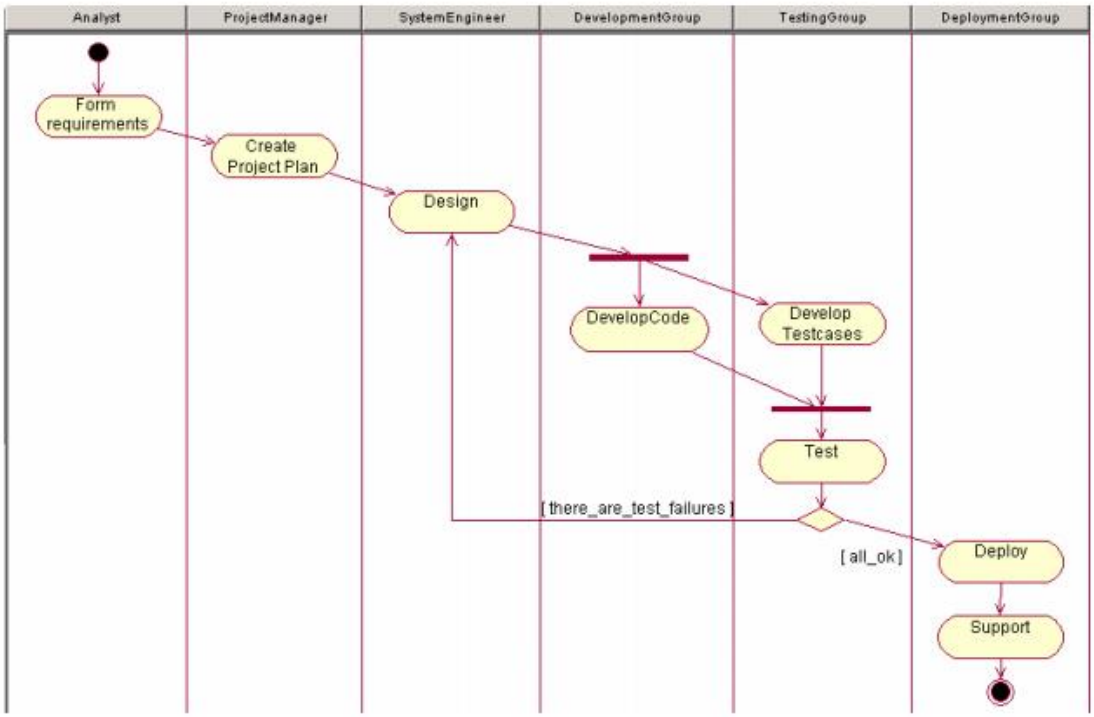

Рисунок 10.5 – Використання діаграми діяльності для відображення бізнес-процесів

В якості swimlanes в даній діаграмі виступають наступні особи (групи осіб):

1) аналітик, який розробляє вимоги до проекту;
2) керівник проекту, який складає план виконання робіт;
3) системний інженер, що проектує систему;
4) група розробників, які створюють програмний код;
5) група тестувальників, які формують варіанти тестування та тестують створену систему;
6) група впровадження, яка поставляє систему кінцевому користувачу та здійснює підтримку.

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями.
2. Побудувати 3 діаграми діяльності для окремих варіантів використання системи.

Вимоги:

1) Кожна діаграма повинна містити не менше 6 діяльностей,
2) При побудові кожної діаграми використовувати стани прийняття рішення та синхронізації.

3. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке діаграма діяльності?
2. Які основні елементи використовуються у діаграмах діяльності?
3. Що таке потік керування у діаграмах діяльності?
4. Яку роль відіграють умови переходу (decision nodes)?
5. У яких випадках застосовуються діаграми діяльності?
6. Чим діаграми діяльності відрізняються від блок-схем?

Лабораторна робота №11. Побудова діаграм класів

Мета: Одержати практичні навики у побудові діаграм класів для обраного варіанту комп'ютерної системи

Теоретичні відомості

Діаграми класів (Class diagrams) – головний тип діаграм UML, які відображають логічну структуру програмної системи та суттєво впливають на процес генерації програмного коду. Основними елементами діаграми класів у Rational Rose є безпосередньо класи (classes) та відношення між ними (relations).

Клас – сукупність логічних об'єктів, які мають схожі характеристики і відрізняються однаковою поведінкою. В ООП характеристики об'єктів певного класу представлені сукупністю атрибутів, а поведінка – сукупністю операцій.

Рисунок 11.1 – Графічне представлення класу на діаграмі класів у Rational Rose

У Rational Rose кожен клас графічно представлений прямокутником, що має три секції: ім'я класу, перелік атрибутів та перелік операцій (рис. 11.1). Для кожного атрибуту задається тип даних, для кожної операції тип даних для значення, що повертається, та перелік параметрів. Атрибути та операції можуть бути визначені для кожного об'єкта класу, чи для класу в цілому (static attributes and operations). Також для всіх атрибутів та операцій можна визначити тип видимості (public, protected, private).

Розрізняють декілька типів класів:

1) Конкретний клас – для даного класу можна створити об'єкт.

2) Абстрактний клас – клас, для якого не можна створити об'єкт. Даний клас виступає чистою абстракцією, від нього можна наслідувати конкретні класи.

3) Параметризований клас – клас, для визначення якого використовується список формальних параметрів, які впливають на атрибути та операції (аналог шаблона в C++). Найчастіше такі класи використовуються для створення абстрактних типів даних {списки, вектори, стеки, черги і т. п.).

4) Інтерфейс – клас, що містить тільки визначення набору операцій (без реалізації). У мові C-H- аналогом даного класу є клас, всі операції якого є

чистими віртуальними функціями. В мові Iaya класи можуть реалізовувати декілька інтерфейсів (інтерфейси використовуються для реалізації концепції множинного наслідування).

Створення нових класів та операцій класі.

У Rational Rose для кожного об'єкта на діаграмах взаємодії існує можливість призначити певний клас (чи створити новий). При цьому в прямокутнику об'єкта дані відображаються в наступному форматі <ім'я об'єкта>:<ім'я класу>. Також кожне повідомлення на діаграмі взаємодії представляє певну операцію класу-приймача. Для кожного повідомлення можна обрати існуючий метод класу – приймача чи створити новий. На рис. 2 наведено фрагмент модифікованої діаграми послідовності для варіанта використання «Переглянути список студентів».

Після створення набору класів з операціями можна відобразити їх на діаграмі класів. При цьому кожен клас буде мати ім'я та перелік операцій, визначених користувачем. На рис. 3 показано відображення класу MainDialog з операціями, створеними для повідомлень 2 та 4 попередньої діаграми (повідомлення 1 буде показано пізніше на загальному вигляді діаграми класів, оскільки воно визначено для класу Dialog який є базовим для даного класу).

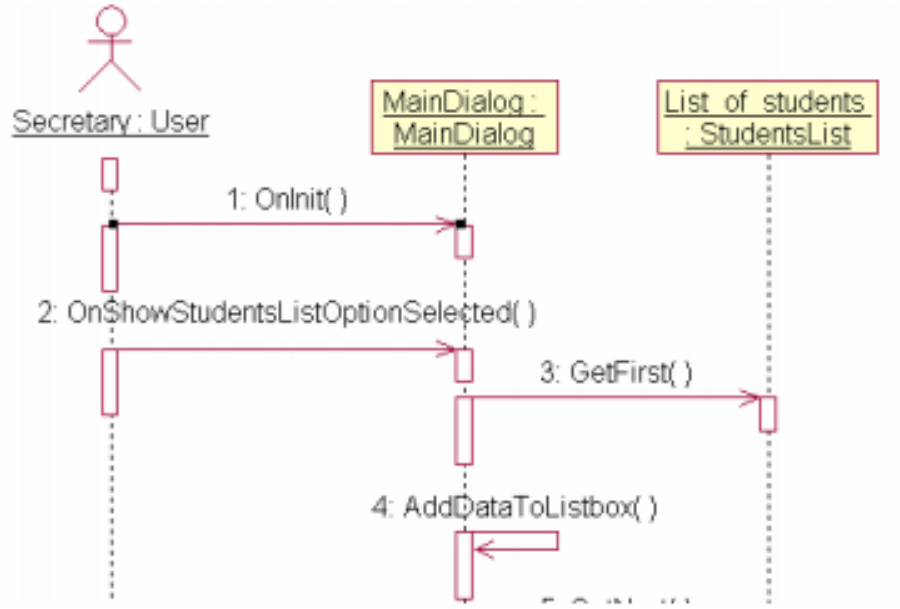

Рисунок 11.2 – Фрагмент діаграми послідовності для варіанта використання «Переглянути список студентів»

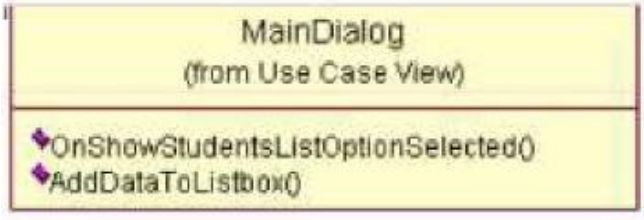

Рисунок 11.3 – Відображення класу MainDialog на діаграмі класів

Також нові класи та операції класів можна створювати безпосередньо з діаграми класів.

Додавання атрибутів класів.

Після створення переліку класів із набором операцій необхідно для кожного класу створити набір атрибутів – даних, якими оперує програма у процесі виконання операцій класу. Наприклад, виходячи з діаграми послідовності, зображеної на рис. 4 можна визначити для класу MainDialog атрибути для кнопки (Button), яку натискає користувач для перегляду списку студентів та списку, до якого безпосередньо додається інформація про кожного студента.

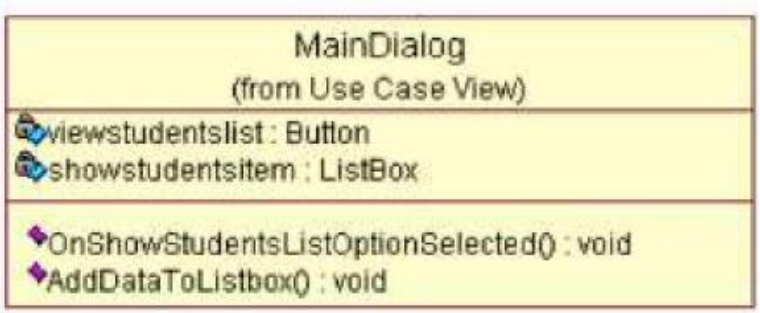

Рисунок 11.4 – Клас MainDialog з повним набором атрибутів та операцій

Повний формат відображення атрибутів на діаграмі класів:

<attribute_name>:<attribute_type>

Повний формат відображення операцій на діаграмі класів:

<operation_name>(<list_of_arguments>):<type_of_return_value>

Відношення між класами.

Rational Rose для нотації Unified можна задати 5 основних типів відношень між класами:

1 Відношення асоціації (assotiation) — визначає абстрактний зв’язок між класами, може представляти аналог відношення «m до n».

Приклад відношення між класами Student та Course (кожен студент відвідує декілька курсів (дисциплін), для кожної дисципліни визначений набір студентів, які її відвідують). Представлення в програмному коді:

```
Class Student
{ …..
Course* theCourses;
……};
Class Course;
{…..
Student* theStudents;
….};
```

При цьому Course* в класі Student може вказувати на масив дисциплін, а

Student;* у класі Course – на перелік студентів.

2 Відношення агрегації (aggregation) – відношення типу «частина - ціле», при якому час життя класу частини не співпадає в часом життя класу цілого.

Приклад: відношення між класами Student та Ticket (у час переоформлення студентського квитка студент не має його).

Представлення у програмному коді:

Class Student

{……

Ticket* ticket;

…..};

Атрибут ticket при цьому може створюватися та видалятися (за допомогою операторів new та delete) в коді будь-яких методів класу Student

3 Відношення композиції (composition) – відношення типу «частина – ціле», при якому час життя класу частини співпадає з часом життя класу цілого.

Приклад: відношення між класами Student та Data_of_Birth (кожен студент у будь-який момент часу характеризується датою народження).

Представлення у програмному коді:

Class Student

{……

Date_of_Birth theDate_of_Birth;

…..};

Атрибут Date_of_Birth існує протягом усього часу життя об'єкта класу Student.

4 Відношення наслідування (generalization) – відношення типу «загальне — часткове»

Приклад: відношення між класами Student та Astudent (кожен студент у будь-який момент часу характеризується датою народження).

Представлення у програмному коді:

Class Student;

Class Astudent : public Student {…..

float GetScholarBonus();

…..};

Клас Astudent має додатковий метод GetScholarBonus(), який розраховує надбавку до стипендії для студента відмінника.

5 Відношення інстанціювання (instantiation) – визначає інстанціювання нового класу з параметризованого класу шляхом підставлення фактичних параметрів у формальні параметри шаблону.

Приклад: відношення між класами List та StudentList (з абстрактного списку інстанціюється список студентів).

Представлення в програмному коді:

template <class T> class List { …. } // List – абстрактний список.

typedef List <Student> StudentsList; // List<Student> – список студентів.

Для класу, що інстанціюється, генерація програмного коду відсутня, оскільки він генерується компілятором автоматично при зверненні до параметризованого класу (підставлення фактичних параметрів у формальні параметри шаблона).

На рис. 5 наведено приклад діаграми класів для системи обліку успішності студентів у деканаті, яка містить перелік класів, пов’язаних різними типами відношень.

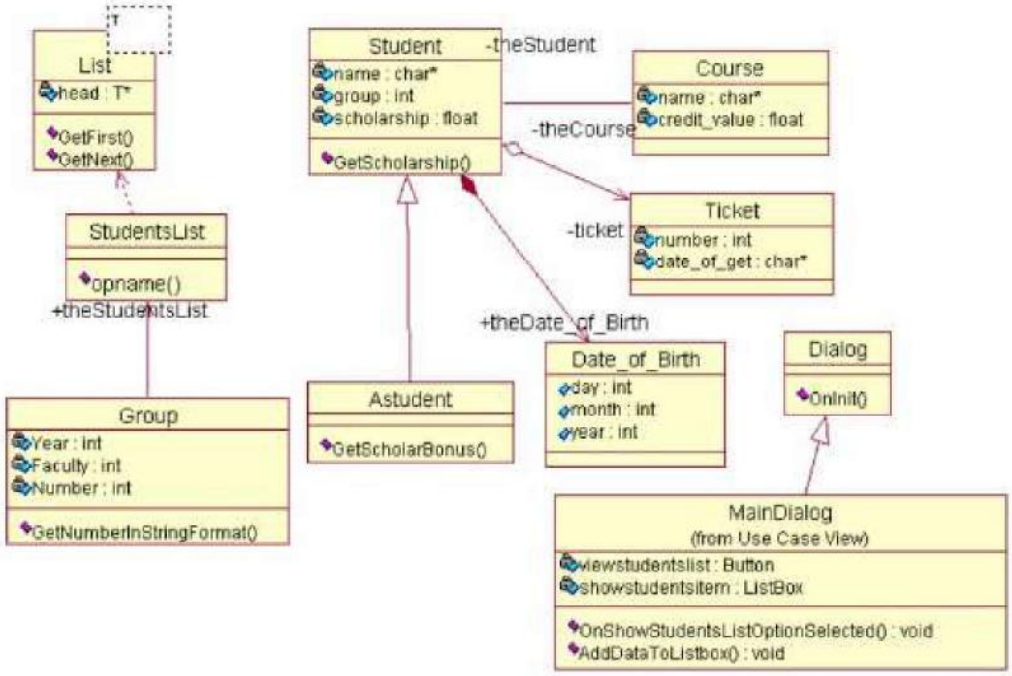

Рисунок 11.5 – Діаграма класів для системи обліку успішності навчання студентів у деканаті

Хід роботи

1. Ознайомитися з літературою та основними теоретичними відомостями.
2. Для всіх об’єктів на діаграмах взаємодії призначити (створити) певний клас; для кожного повідомлення призначити (створити) відповідний метод (операцію) для класу об’єкта-приймача.
3. Розташувати створені класи з переліком операцій на діаграмі класів.
4. Для кожної операції визначити атрибути, які вона використовує та при необхідності додати їх до списку атрибутів класу.

5. Для кожного атрибуту задати логічний тип даних, для кожної операції логічний тип даних для return value та для переліку аргументів, якщо вони присутні.

6. Пов’язати класи на діаграмі класів, використовуючи різні типи відношень (асоціацію, агрегацію, композицію, наслідування, інстанціювання).

Вимоги:

1. Діаграма класів повинна містити не менше 10 класів.
2. Для кожного класу визначити не менше 5 атрибутів та 5 операцій.
3. По можливості використати всі типи відношень між класами.

7. Оформити звіт з лабораторної роботи.

Звіт повинен містити:

1. Титульний аркуш.
2. Мету роботи.
3. Покроковий опис виконання завдання.
4. Скріншоти з результатами виконаної роботи (за умови використання ПК при виконанні роботи).

Контрольні запитання

1. Що таке діаграма класів у UML?
2. Які основні елементи діаграми класів?
3. Що таке атрибути та методи класу?
4. Які типи зв'язків існують між класами?
5. Що таке наслідування та асоціація?
6. Яку роль відіграють діаграми класів у проєктуванні програмного забезпечення?

Список використаної літератури

***Основна*:**

1. *A Contemporary Look at Methodological Shifts and Publication Trends in the Business Informatics Community* [Електронний ресурс] // *Business & Information Systems Engineering*, 2025. – Режим доступу: https://link.springer.com/article/10.1007/s12599-025-00934-6.

2. Das Gupta, R., Rahman, A., Showmick, M. I. H. *Exploring the Convergence of HCI and Evolving Technologies in Information Systems* [Електронний ресурс] // *arXiv preprint arXiv:2506.08549*, 2025. – Режим доступу: https://arxiv.org/abs/2506.08549.

3. Goedegebuure, A., Kumara, I., et al. *Data Mesh: a Systematic Gray Literature Review* [Електронний ресурс] // *arXiv preprint arXiv:2304.01062*, 2023. – Режим доступу: https://arxiv.org/abs/2304.01062.

4. Heiland, L., Hauser, M., Bogner, J. *Design Patterns for AI-based Systems: A Multivocal Literature Review and Pattern Repository* [Електронний ресурс] // *arXiv preprint arXiv:2303.13173*, 2023. – Режим доступу: https://arxiv.org/abs/2303.13173.

5. Őri, D., Szabó, Z. A. *A Systematic Literature Review on Business-IT Misalignment Research* [Електронний ресурс] // *Information Systems Frontiers*, 2024. – Режим доступу: https://link.springer.com/article/10.1007/s10257-023-00664-w.

6. Puspitasari, D. *The Strategic Role of Information Systems in Advancing Sustainable Business Practices* [Електронний ресурс] // *International Journal of Data Science and Information Processing*, 2024. – Режим доступу: https://journal.idscipub.com/data/article/view/722

7. Авраменко В. С., Авраменко А. С. *Проєктування інформаційних систем* : навчальний посібник. – Черкаси : ЧНУ ім. Б. Хмельницького, 2017. – 433 с.

8. Лєнков С. В., Грищак О. М., Охрамович М. М., Сотніков Є. О. *Аналіз сучасних та традиційних методів проєктування і виробництва військової техніки з використанням автоматизованих інформаційних систем* // *Збірник наукових праць Центру воєнно-стратегічних досліджень НУОУ ім. І. Черняховського*. – 2024. – № 2. – С. 56–65.

9. Литвин В. В., Шаховська Н. Б. *Проєктування інформаційних систем* : навч. посіб. – Львів : Видавництво Львівської політехніки, 2015. – 412 с.

10. Лук’янчук Ю. А., Тулашвілі Ю. Й. *Проєктування інформаційних систем* : методичні вказівки. – Луцьк : Луцький НТУ, 2021. – 64 с.

11. Маланчук О., Яценко І., Гуменюк І. *Архітектура інтелектуальної інформаційної системи прогнозування складових медичних проектів* // *Applied Aspects of Information Technology*. – 2023. – № 6(2). – С. 95–103. – DOI: 10.15276/aait.2023.02.07.

12. Паращук Л. Я., Паращук С. М. *Рекомендації стосовно використання інформаційних систем для покращення ситуаційної обізнаності органів військового управління* // *Scientific Innovations and Advanced Technologies*. – 2025. – № 2(28). – С. 33–42. – DOI: 10.52058/2786-5274-2025-2(28)-33-42

13. *Проєктування інформаційних систем: Загальні питання теорії проектування ІС* : конспект лекцій / КПІ ім. Ігоря Сікорського. – Київ : КПІ, 2020. – 58 с.

Додаткова:

14. K. Bortnyk, N. Bahniuk, I. Kondius, K. Melnyk, Y. Melnychuk and K. Kondius, "Effective Content Moderation Using Modern AI Tools," 2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2024, pp. 1-8, doi: 10.1109/DESSERT65323.2024.11122238. (Scopus) https://ieeexplore.ieee.org/document/11122238

15. L. Markina, B. Palchevskyi, R. Hrudetskyi, O. Smoliankin, Y. Melnychuk and N. Khrystynets, "Optimization of Ethanol Production Using State-Space Modeling and Optimal Control Technology", Publisher: IEEE (Institute of Electrical and Electronics Engineers Inc.) Proceedings of the IEEE - 2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2023, pp. 1-7, doi: 10.1109/DESSERT61349.2023.10416529 / https://ieeexplore.ieee.org/document/10416529 (наукометрична база Scopus).

16. V. Satsyk, O. Reshetylo, L. Markina, N. Khrystynets, N. Bahniuk and Y. Melnychuk, "Modeling of Improved Solar Energy Installation for Efficient Power Systems," 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), Ceske Budejovice, Czech Republic, 2024, pp. 688-694, doi: 10.1109/ACIT62333.2024.10712532. https://ieeexplore.ieee.org/document/10712532 (наукометрична база Scopus).

17. Y. Melnychuk, V.Satsyk, R.Grudetsky, O.Kuzmych, N.Bahniuk, L.Hlynchuk Reduction of Server Load by Means of CMS Drupal // IEEExplore Digital Library (Scopus), Published in: 2020 10th International Conference on Advanced Computer Information Technologies (ACIT). DOI: 10.1109/ACIT49673.2S20.9208874, ISBN: 978-1-7281-6760-2.

18. Yulia Melnychuk, Oleg Barabash, Oleksandr Laptiev, Valentyn Sobchuk, Ivanna Salanda,, Valerii Lishchyna. Comprehensive Methods of Evaluation of Distance Learning System Functioning. International Journal of Computer Network and Information Security (IJCNIS), Vol. 13, No. 3, June. 2021, pp. 62 – 71. DOI: 10.5815/ijcnis.2021.03.06. (Scopus) http://www.mecs-press.org/ijcnis/v13n3.html.

19. Yuliia Melnychuk, Liudmyla Mialkovska, Halyna Herasymchuk, Iryna Sushyk, Yaroslava Martyniuk, Olena Haponchuk. Management models and methods in modern education: information technologies, sustainability and development. The Global Development of Innovative Technologies and their Impact on the Education, Vol. 16 No. se2 (2023) https://brajets.com/index.php/brajets/article/view/1256 (наукометрична база WoS).

20. Мельничук Ю., Полухтович Т. Цінність знань у розвитку особистості /Т. Полухтович, Ю.Мельничук// Науковий журнал «Молодь і ринок» (Категорія «Б»), індексується: Google Scholar, Polish Scholarly Bibliography, Index Copernicus. Дрогобич. №1 (187) ,2021. С.100-103. (фахове видання, наукометрична база Index Copernicus).

21. Мельничук Ю., Полухтович Т. (2025) Хмарні технології та онлайн-сервіси: вплив на цифрову трансформацію суспільства / Журнал «Наукові інновації та передові технології». Серія «Педагогіка» №4(44). Вид. група «Наукові перспективи». Київ, 2025. С.1311-1321. DOI: https://doi.org/10.52058/3041-1572-2025-2(10)-303-315 (фахове видання).

22. Мельничук Ю.Є. Алгоритми функціонування інформаційних систем освітнього призначення // «Наука і техніка сьогодні» (Серія «Педагогіка», Серія «Право», Серія «Економіка», Серія «Фізико-математичні науки», Серія «Техніка»)»: журнал. 2023. № 3(17), с. 576-585 (фахове видання).

23. Мельничук Ю.Є. Інтелектуальні технології в освіті. – Луцьк: Терен, 2023. – 241 с. (Затверджено вченою радою ЛНТУ, витяг з протоколу №8 від 28 березня 2023 року).

24. Мельничук Ю.Є. Інформаційний менеджмент як складова підготовки майбутніх фахівців у закладах вищої освіти. // Журнал «Перспективи та інновації науки» (Серія «Педагогіка», Серія «Психологія», Серія «Медицина»), № 8(42), 2024. С. 407-419 (фахове видання).

25. Мельничук Ю.Є. Принципи побудови інформаційних систем освітнього призначення. // «Комп'ютерно-інтегровані технології: освіта, наука, виробництво»: журнал. 2023. № 50, с. 77-84 (фахове видання).

26. Мельничук Ю.Є. Розвиток алгоритмів створення веб-орієнтованих інформаційних систем // «Наука і техніка сьогодні» (Серія «Педагогіка», Серія «Право», Серія «Економіка», Серія «Фізико-математичні науки», Серія «Техніка»)»: журнал. 2023. № 2(16) 2023. С. 392-400 (фахове видання).

27. Мельничук Ю.Є., Редько О.І., Сушик О.Г. (2025) Інтерфейси інформаційних систем у професійній підготовці майбутніх фахівців з інформатики / Журнал «Наука і техніка сьогодні» (Серія «Педагогіка», Серія «Право», Серія «Економіка», Серія «Фізико-математичні науки», Серія

«Техніка»)»: журнал. 2025. № 8(49) 2025. С. 829-840. DOI: https://doi.org/10.52058/2786-6025-2025-8(49)

28. Мельничук Ю.Є., Редько О.І., Сушик О.Г. Stem-підхід під час викладання дисциплін ІТ-профілю. Перспективи та інновації науки (Серія «Педагогіка», Серія «Психологія», Серія «Медицина»)»: журнал. 2024. № 11 (45) 2024. С. 648-658 (фахове видання).

29. Мельничук Ю.Є., Сокотов Ю.В., Погрібняк М.Ю. Роль ІКТ у підготовці здобувачів професійної освіти // Журнал «Перспективи та інновації науки» (Серія «Педагогіка», Серія «Психологія», Серія «Медицина»), № 4(9), 2022. С. 231-242. (фахове видання).

Проектування інформаційних систем [Текст]: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання/ уклад. Ю.Є. Мельничук Луцьк: ЛНТУ, 2026. 60 с.

Комп'ютерний набір Ю.Є. Мельничук
Редактор Ю.Є. Мельничук

Підп. до друку «__»_____2026 р. Формат 60х84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 3.
Тираж 50 прим.

Луцького національного технічного університету
43018, м. Луцьк, вул. Львівська, 75