Co-funded by the
Erasmus+ Programme
of the European Union

Г.В. Табунщик, Т.І. Каплієнко, О.А Петрова,
О.В. Шитікова

ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ ІНФРАСТРУКТУРИ МЕДИЧНИХ ТА ТЕЛЕМЕДИЧНИХ СИСТЕМ

НАВЧАЛЬНИЙ ПОСІБНИК

Житомир
Видавець ПП "Євро-Волинь"
2021

УДК 004.41:61 (075.8)
П79

Рекомендовано як навчальний посібник для студентів напрямків «Телемедичні та медичні системи» та «Комп'ютені науки» на засіданні Вченої Ради Національного університету «Запорізька Політехніка» від 02.03.2020 р., протокол №4/20.

Р е ц е н з е н т и :

С. І. Гоменюк *– доктор технічних наук, професор, декан математичного факультету Запорізького національного університету*

О.Ф. Тарасов *– доктор технічних наук, професор, завідувач кафедри комп'ютерних інформаційних технологій Донбаської державної машинобудівної академії*

Видання здійснено за підтримки міжнародного проєкту «Інноваційна мультидисциплінарна навчальна програма для підготовки бакалаврів та магістрів зі штучних імплантів для біоінженерії» (586114-EPP-1-2017-1-ES-EPPKA2-CBHE-JP), що фінансується Європейською Комісією.

Підтримка Європейською комісією випуску цієї публікації не означає схвалення змісту, який відображає лише думки авторів, і Комісія не може нести відповідальність за будь-яке використання інформації, що міститься в ній.

Табунщик Г. В.

П79 Проєктування інформаційної інфраструктури медичних та телемедичних систем / Г. В. Табунщик, Т.І. Каплієнко, О.А Петрова, О.В. Шитікова. Вид. ПП "Євро-Волинь", Житомир, – 2021. 198 с.

ISBN 978-617-7992-10-2

Навчальний посібник містить підходи та засоби аналізу, проєктування та моделювання програмного забезпечення медичних та телемедичних систем.

Видання призначене для студентів комп'ютерних спеціальностей вищих навчальних закладів, а також може бути використаним аспірантами, науковими та педагогічними працівниками, фахівцями-практиками.

Навчальний посібник видано за підтримки проєкту Інноваційна мультидисциплінарна навчальна програма для підготовки бакалаврів та магістрів зі штучних імплантів для біоінженерії 586114-EPP-1-2017-1-ES-EPPKA2-CBHE-JP [BIOART].

Проєкт фінансується при підтримці Європейської комісії. Зміст матеріалу відображає думку авторів та Європейська комісія не несе відповідальності за використання інформації, що міститься в навчальному посібнику.

УДК 004.41 (075.8)

ISBN 978-617-7992-10-2

ЗМІСТ

ВСТУП

Інтернет медичних речей (IoMT), підключена до людини інфраструктура медичних пристроїв, програмні додатки в системі охорони здоров'я та послуг значно трансформує медичний сектор. Цей стрімкий розвиток інформаційних технологій передбачає наявність у молодих фахівців знань та навичок з проєктування, моделювання та аналізу програмного забезпечення цих систем.

У цьому навчальному посібнику автори постарались найбільш органічно викласти матеріал, щоб підвищити рівень його засвоєння студентами напрямів «Телемедичні та медичні системи» та «Комп'ютерні науки». Викладений матеріал був апробований під час читання курсів «Медична інформаційна інфраструктура», Комп'ютерні системи управління проєктами, регуляція та стандартизація в медичній галузі», що вивчається студентами за освітньо-професійною програмою «Телемедичні та біомедичні системи» та «Проєктування та моделювання IoT систем», що вивчається студентами за спеціальністю «Комп'ютерні науки».

Метою книги є надання систематичного огляду до процесів розроблення складних інформаційних систем з використанням сучасних підходів, мов програмування та мови моделювання UML.

Структурно підручник містить дві основні частини. Перша частина присвячена основним поняттям медичних інформаційних систем Наведені види і класифікація медичних інформаційних систем, методи керування ризиками в медичних системах та розглянуто питання верифікацій біомедичних систем. Друга частина повністю містить методологічні основи аналізу проєктування та моделювання інформаційних систем.

Видання орієнтоване на студентів комп'ютерних спеціальностей закладів вищої освіти, а також може використовуватися аспірантами, науковими та педагогічними працівниками, практичними фахівцями.

Матеріал рукопису відповідає змісту програми дисципліни матеріалів лекцій, що викладається в Національному університеті «Запорізька політехніка». Зокрема, авторами використано матеріали, отримані при виконанні проєкту Інноваційна

мультидисциплінарна навчальна програма для підготовки бакалаврів та магістрів зі штучних імплантів для біоінженерії

Терміни, визначення яких наводяться в тексті виділено жирним курсивом. Кожний розділ закінчується контрольними запитаннями та практичними завданнями, що можуть бути використані як для самоперевірки, так і при підготовці лабораторних, практичних та контрольних завдань.

Для підвищення наочності матеріалу в тексті використовуються наступні позначення:

 – визначення

 – практичні завдання

 – приклади, контрольні запитання

 – рекомендована література

ЧАСТИНА 1. ОСНОВНІ ПОНЯННЯ З РОЗРОБЛЕННЯ МЕДИЧНИХ СИСТЕМ

1. МЕДИЧНІ ІНФОРМАЦІЙНІ СИСТЕМИ

1.1 Стан інформатизації системи охорони здоров'я

Сучасною тенденцією у багатьох країнах є реформа системи охорони здоров'я, що полягає в першу чергу у створенні сучасної електронної системи, що дозволить значно підвищити ефективність та прозорість охорони здоров'я.

Стрімкий розвиток інформаційно-комунікаційних технологій (ІКТ) та цифрова трансформація може принести істотну вигоду як громадській охороні здоров'я, так й індивідуальному медичному обслуговуванню, адаптуючи способи надання медичних послуг і характер управління системами охорони здоров'я на всіх рівнях.

Основні установи Організації Об'єднаних Націй (ООН) з питань охорони здоров'я та телекомунікацій, Всесвітня організація охорони здоров'я (ВООЗ) та Міжнародний союз електрозв'язку (МСЕ) визнали важливість співпраці в питаннях електронної охорони здоров'я, закликаючи країни розробляти національні стратегії.

Багато електронних систем охорони здоров'я характеризуються відсутністю сумісності інформаційних систем у сфері охорони здоров'я, відсутністю єдиного унікального ідентифікатора для пацієнтів, недосконалістю інформаційної інфраструктури та взаємодії між загальнодержавними реєстрами, недосконалістю ряду реєстрів, недостатністю фахових спеціалістів для автоматизації та управління змінами, недостатністю комп'ютерного та мережевого обладнання в закладах охорони здоров'я тощо.

Рівень комп'ютеризації, покриття медичними інформаційними системами надавачів послуг в сфері охорони

здоров’я залишається низьким. Зазначена обставина з однієї сторони є викликом, а з іншої сторони – породжує можливості для швидкого розвитку інформатизації, оскільки відсутня необхідність переробляти історично накопичені інформаційні системи, є можливість відразу впроваджувати найновіші ІКТ.

1.1.1 Цифрові трансформації в медичній галузі

Цифрова трансформація в Україні в медичній галузі почалась в 2018 році. Була створена робоча група з цифрової трансформації охорони здоров’я, основним завданням якої є сприяння актуальним змінам у галузі на основі широкого використання цифрових рішень, які забезпечують необхідні засоби для зміцнення здоров’я, запобігання захворювань та надання комплексних послуг, що базуються на потребах людей. Одним з пріоритетних напрямів є створення цифрової екосистеми надання медичних послуг.

Враховуючи нову стратегію керівництва України — «держава у смартфоні», правильним кроком у розвитку галузі охорони здоров’я стане розробка стратегії інтегрованої охорони здоров’я з впровадженням цифрових технологій та цифрових пристроїв у парадигму здоров’я людей. Цей процес має бути підкріплений розробкою регуляторних вимог щодо кібербезпеки медичних пристроїв, як компонентів критичної інфраструктури.

Цифровізація (з англ. digitalization) — це впровадження цифрових технологій в усі сфери життя: від взаємодії між людьми до промислових виробництв х, від предметів побуту до дитячих іграшок, одягу тощо [1]. Це перехід біологічних та фізичних систем до кібербіологічних та кіберфізичних (об'єднання фізичних та обчислювальних компонентів). Перехід діяльності з реального світу у світ віртуальний (онлайн).

Цифрова трансформація (Digital transformation) - це трансформація бізнесу шляхом перегляду бізнес-стратегії або цифрової стратегії, моделей, операцій, продуктів, маркетингового підходу, цілей тощо, шляхом прийняття цифрових технологій.

Цифрові технології: Інтернет речей, роботизація та кіберсистеми, штучний інтелект, великі дані, безпаперові технології, адитивні технології (3D-друк), хмарні та туманні обчислення, безпілотні та мобільні технології, біометричні, квантові технології, технології ідентифікації, блокчейн тощо.

В Україні створено Міністерство цифрової трансформації України. Основними видами діяльності якого є:

– формування та реалізація державної політики у сфері цифровізації, цифрової економіки, цифрових інновацій, електронного урядування та електронної демократії, розвитку інформаційного суспільства;

– формування та реалізація державної політики у сфері розвитку цифрових навичок та цифрових прав громадян;

– формування та реалізація державної політики у сфері відкритих даних, розвитку національних електронних інформаційних ресурсів та інтероперабельності, розвитку інфраструктури широкосмугового доступу до Інтернету та телекомунікацій, електронної комерції та бізнесу;

– формування та реалізація державної політики у сфері надання електронних та адміністративних послуг;

– формування та реалізація державної політики у сфері електронних довірчих послуг;

– формування та реалізація державної політики у сфері розвитку ІТ-індустрії;

– виконання функцій центрального засвідчувального органу шляхом забезпечення створення умов для функціонування суб’єктів правових відносин у сфері електронних довірчих послуг.

1.1.2 Інтернет медичних речей

Інтернет медичних речей (IoMT) - це термін, що перекриває безліч застосувань. Термін IoMT добре підходить для задоволення потреб галузі охорони здоров'я, підтримуючи перехід від неуспішної допомоги до координованої допомоги та реактивних до проактивних підходів до надання медичної допомоги, наприклад. Використання цієї тенденції за допомогою правильних застосувань, для правильних клієнтів з правильними партнерами та відповідними бізнес-моделями має вирішальне значення для зацікавлених сторін у галузі охорони здоров'я, щоб вижити в умовах жорсткої конкуренції, яку підтримують і стартапи, і технічні гіганти.

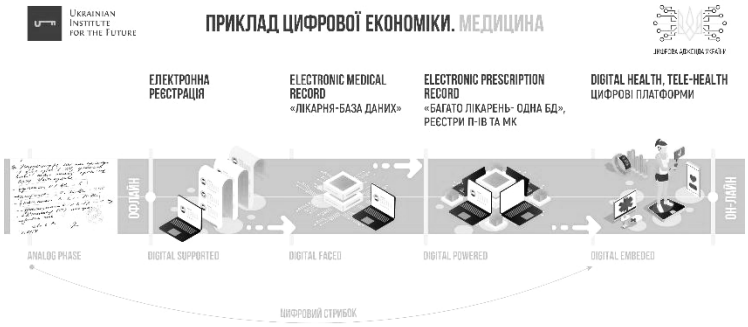


Рисунок 1.1 - Приклад цифрової економіки. Медицина [1]

На сьогоднішній день існують наступні основні напрямки, що сприяють впровадженню медичного інтернету речей:

– пристрої та мобільні додатки для віддаленого спостереження за станом здоров'я - ці пристрої записують дані в медичні карти пацієнтів в реальному часі, проводять аналіз і відправляють повідомлення з рекомендаціями постачальникам і пацієнтам;

– носимі пристрої - гаджети, які безперервно відстежують повсякденну активність пацієнтів і повідомляють такі відомості, як кількість кроків, витрачені калорії, серцевий ритм і т. д., допомагаючи попередити і, можливо, навіть запобігти виникненню станів, що вимагають надання термінової медичної допомоги;

– пацієнтоорієнтована медицина містить пристрої, що дозволяють надавати медичну допомогу з урахуванням індивідуальних переваг або потреб пацієнтів.

Цільову аудиторію користувачів медичних виробів та інших пристроїв, що працюють за технологією «IoMT», можна також розділити на 3 групи.

Медичні працівники - використовують технології «інтернету медичних речей», в основному, для експрес-діагностики на місці надання медичної допомоги.

Пацієнти - особи, які страждають певними захворюваннями, або що входять до групи ризику; але також і умовно здорові особи, які бажають контролювати свій спосіб життя.

Професіонали - спортсмени, співробітники служби охорони, військові, пожежники, працівники віддалених бурових станцій, водії.

Технології «Інтернету медичних речей» можна класифікувати відповідно від місця знаходження під час використання. Відповідно, можна виділити наступні групи.

Фіксовані - прилади, які мають конкретну точку локалізації в просторі. Як правило, це продиктовано не технічними обмеженнями, а необхідністю отримання інформації з сенсорів в конкретній зоні, локації. Прикладом можуть служити прилади, що збирають інформацію про навколишнє середовище (термометри, барометри, дозиметри і т.д.) або провідні відео спостереження.

Портативні: компактні, легко переносяться (до місця знаходження пацієнта) прилади, які використовуються на вимогу.

Наприклад, тонометри, глюкометри, ваги, ЕКГ, портативні аналізатори і т.д.

Носибельні: прилади, які тривалий час або постійно перебувають на тілі пацієнта (умовно здорового особи) з метою проактивного моніторингу фізіологічних функцій. Наприклад, фітнес-трекери, ЕКГ-датчики, вбудовані в одяг що носиться, лінзи, здатні в слізної рідини визначати рівень глюкози і т.д.

Імплантовані: вживлені в тіло людини прилади, які здатні передавати з сенсорів якусь інформацію про стан здоров'я і, отримуючи «зворотний зв'язок», коректувати виникаючі зміни в організмі. Певним прикладом можуть служити інсулінові помпи.

1.2 Види та класифікація медичних інформаційних систем

1.2.1 Основні визначення

У сучасному суспільстві автоматизація та інформатизація медицини - процес неминучий. Величезні обсяги медичної інформації, сувора звітність і її стандартизація, високі вимоги до якості послуг - все це веде до необхідності автоматичної обробки та електронного зберігання даних. Щоб всю цю інформацію було зручно обробляти, зберігати і використовувати, багато лікарень і центри здоров'я вже перейшли на електронний документообіг, інші - готуються до його впровадження.

Ключовою ланкою інформатизації охорони здоров'я є медичні інформаційні інфраструктури та системи [2-9].

Медична інформаційна система (МІС) - це сукупність інформаційних, організаційних, програмних і технічних засобів, призначених для автоматизації медичних процесів.

Перш за все, система МІС призначена для автоматизації клінічних напрямків роботи медичних працівників - впроваджує

електронні медичні картки пацієнта, організовує ефективну комунікацію між усіма учасниками лікувально-діагностичного процесу.

Медичні інформаційні системи полегшили роботу лікаря по веденню прийому пацієнта, заповнення медичних карт, забезпечили повноцінну роботу з талонами пацієнтів, завдяки величезній базі даних пацієнтів, що перебувають на лікуванні в установі, дозволили збирати масу статистичних звітів, на створення яких раніше йшло б величезну кількість часу і людських трудовитрат, реорганізувати багато підрозділів, а також кардинально змінити всі що протікають в установі бізнес-процеси.

Медична інформаційна система - це основа для побудови єдиного добре структурованого інформаційного простору медичного закладу, в якому комфортно і співробітникам, і пацієнтам. Без інформаційної системи неможливо ефективно використовувати лабораторне та діагностичне обладнання, оперативно отримати довідкову інформацію, коректно і своєчасно виставляти рахунки в страхову компанію, переробляти весь той величезний обсяг інформації, який обрушується на будь-якого співробітника клініки, від молодшого медперсоналу до головного лікаря. Побудова на єдиній базі даних і наявність спеціальних інструментів інтеграції з зовнішніми інформаційними системами формує єдиний інформаційний простір, який забезпечує користувачу доступ до повних, оперативним і достовірним даними. Своєчасне отримання інформації, скорочення термінів обстеження, контроль над виконанням медичних стандартів - все це дозволяє підвищити якість лікування пацієнта.

1.2.2 Класифікація медичних інформаційних систем

Класифікація МІС заснована на ієрархічному принципі і відповідає багаторівневій структурі охорони здоров'я:

- базовий рівень (МІС для лікарів різного профілю);

- рівень установ (МІС лікувально профілактичних установ ЛПУ - поліклінік, стаціонарів, диспансерів, лікарень швидкої допомоги);

- територіальний рівень (МІС для профільних і спеціалізованих медичних служб і регіональних органів управління охороною здоров'я);

- державний рівень (МІС Міністерства охорони здоров'я).

Медичні інформаційні системи базового рівня представлені системами інформаційної підтримки технологічних процесів (медико-технологічні ІС). Системи цього класу призначені для інформаційного забезпечення прийняття рішень у професійній діяльності лікарів різних спеціальностей. Основна їх мета - комп'ютерна підтримка роботи лікаря в клініці, гігієніста, лаборанта та ін. Вони дозволяють підвищити якість профілактичної та лікувально-діагностичної роботи, особливо в умовах масового обслуговування при дефіциті часу і кваліфікованих спеціалістів. За важливість справ медико-технологічні ІС можна розділити на наступні групи:

- медичні інформаційно-довідкові системи;

- медичні консультативно-діагностичні системи;

- медичні приладо-комп'ютерні системи;

- автоматизоване робоче місце лікаря.

Рисунок 1.2 – МІС базового рівня

Медичні інформаційно-довідкові системи призначені для пошуку і видачі медичної інформації за запитом користувача.

Інформаційні масиви таких систем містять медичну довідкову інформацію різного характеру. Необхідність накопичення великих обсягів професійної інформації та оперування з ними - одна з проблем, з якою приходиться стикатися лікарям. Інформаційно-довідкові системи поліпшують рішення цієї проблеми, виступаючи як засіб надійного зберігання професійних знань, що забезпечує зручний і швидкий пошук необхідних відомостей. Системи цього класу не здійснюють обробку інформації.

Інформаційні масиви таких систем, як правило, проблемно орієнтовані: це і наукова інформація з різних медичних дисциплін, і довідкова статистична і технологічна інформація широкого профілю і обліково-документальна інформація.

Інформаційно-довідкові системи підрозділяються за видами збереженої інформації (клінічна, наукова, нормативно-правова та ін.), по її характеру (первинна, вторинна, оперативна, оглядово-аналітична, експертна, прогностична і т. п.), по об'єктового ознакою (матеріально-технічна база, лікарські засоби та інше).

Медичні консультативно-діагностичні системи призначені для діагностики патологічних станів при захворюваннях різного профілю і для різних категорій хворих, в тому числі на основі інтелектуального (експертного) підходу. В епоху масового впровадження персональних комп'ютерів в усі сфери сучасного життя природним є прагнення використовувати комп'ютерні системи для підтримки все більш складних видів людської діяльності. Однією з них є діяльність лікаря, ключовий пункт роботи якого - прийняття діагностичних і лікувальних рішень. Оскільки прийняття рішень є результатом переробки певної інформації про пацієнта і базується на використанні накопичених знань, розробляються комп'ютерні системи штучного інтелекту і, зокрема, експертні системи (або системи, засновані на знаннях), здатні допомогти лікарю в рішенні задач діагностики і вибору тактики лікування. Типова інтелектуальна система будується за певною схемою.

Основна частина системи - процесор логічного виводу - відповідає за функціонування системи і проведення міркувань. Робоча пам'ять містить інформацію про розв'язувані задачі, зазвичай у формі об'єкт - атрибут -значення (наприклад, пацієнт - температура - 37С або пацієнт - діагноз -доброякісна гіперплазія передміхурової залози). База знань містить знання про предметну область, зазвичай представлені у вигляді правил, ЯКЩО-ТО. Процесор виведення шукає правила в базі знань, які можуть бути застосовані для пошуку нових даних на підставі поточних фактів з робочий пам'яті, таким чином, на кожному кроці виведення робоча пам'ять поповнюється новими відомостями. Такий процес триває до тих пір, доки не буде отримана необхідна інформація. Область

застосування експертних систем - це завдання, як правило, вузькоспеціалізовані, для яких відсутній чітко формовані алгоритми рішень, дані завдання нечіткі і часто неточні (тобто система повинна працювати в умовах невизначеності).

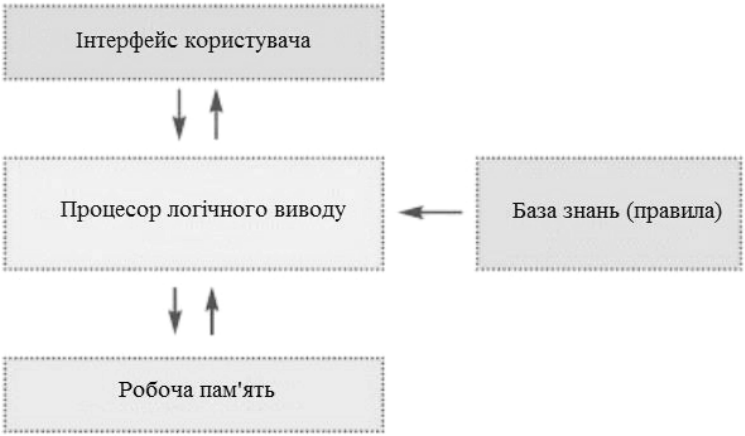


Рисунок 1.3 - Основні складові інтелектуальної системи

За описаним принципом будуються багато медичних системи обробки інформації та експертного аналізу. Дані системи дозволяють на основі інформації про клінічні прояви і тестів зробити в короткий термін кваліфікований висновок по кожному конкретному випадку і відпрацювати оптимальну стратегію лікування хворого.

Медичні приладо-комп'ютерні системи призначені для інформаційної підтримки і / або автоматизації діагностичного і лікувального процесу, що здійснюються при безпосередньому контакті з організмом хворого (наприклад, при проведенні реєстрації фізіологічних параметрів).

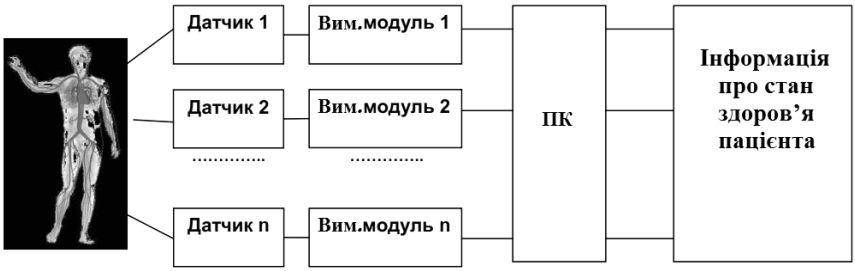


Рисунок 1.4 - Принцип функціонування медичної приладо-комп'ютерної системи

Медичні інформаційні системи рівня лікувально-профілактичних установ. Це МІС, засновані на об'єднанні всіх інформаційних потоків ЛПУ в єдину систему і забезпечують автоматизацію різних видів діяльності установи. Найчастіше така інформаційна система створюється поетапно, причому так, щоб черговий інформаційний блок її приносив конкретну користь і був основою для наступного блоку.

Рисунок 1.5 – МІС рівня лікувально-профілактичних установ

Медичні інформаційні системи територіального рівня. Це програмні комплекси, що забезпечують роботу автоматизованих систем управління органів

охорони здоров'я на рівні території (міста, області), включаючи управління спеціалізованими та профільними медичними службами:

1. Адміністративно-управлінські підсистеми МІС підготовлені для виконання організаційних завдань, що вирішуються керівниками територіальних медичних служб (обласних, міських, районних відділів охорони здоров'я).

2. МІС для науково-дослідних інститутів та вищих навчальних закладів вирішують три основні завдання: інформатизацію процесу навчання, науково-дослідної роботи.

3. Статистичні інформаційні системи, які здійснюють збір, обробку та отримання по території (обласних, міських, районних відділів охорони здоров'я) зведені дані по основним медико-соціальними показниками.

4. Комп'ютерні телекомунікаційні медичні мережі, що забезпечують створення єдиного інформаційного простору на рівні регіону.

Державний рівень. Автоматизовані системи управління (АСУ МОЗ), призначені для інформаційної підтримки служб державного рівня МОЗ.

Також МІС можна класифікувати за ступенем автоматизації процесів збору інформації, за типом інформаційної бази, за видом розв’язання задач рис.1.6.

Рисунок 1.6. – Класифікація МІС

1.2.3 Структура медичних інформаційних систем

МІС - це система роботи з медичною інформацією, що надходить в реальному часі або зберігається в базі даних. Без бази даних не може обійтися жодна інформаційна система. Наявність бази даних дозволяє застосовувати стандартні процедури обробки файлів. МІС, як і кожен її автономний блок, складається з обов'язкових програмних модулів:

1. Модуль збору інформації.
2. База даних.
3. Модуль обробки і аналізу даних.
4. Модуль управління документообігом.
5. Модуль управління медичними апаратними засобами.

Сучасні МІС складаються з окремих блоків (підсистем), здатних працювати автономно або в сукупності з іншими блоками через локальні або глобальні мережі. Персональні комп'ютери (ПК) користувачів даної МІС (від одного до декількох) утворює

автоматизовані робочі місця (АРМ). За допомогою АРМ відбувається організація доступу до інформаційної системи для кожного медичного працівника (лікаря, фельдшера, лаборанта, медичної сестри). Будь-яке АРМ спеціалізоване для виконання строго конкретних професійних завдань і роботи з встановленої медичною документацією. Забезпечує роботу всіх блоків головний комп'ютер (центральний), який відповідає за виконання найбільш важливих і відповідальних операцій і таким чином є його основним обчислювальним ресурсом. Система забезпечує інформаційну підтримку всіх служб медичного закладу - від документообігу та фінансового обліку до ведення клінічних записів про пацієнта, інтеграції з медичним обладнанням та підтримки прийняття рішень.

Основні підсистеми:

1. Реєстраційно-статистична підсистема. Вона дозволяє реєструвати всі події та факти, що відбуваються в лікувальному закладі. Така підсистема скорочує рутинну роботу персоналу, допомагає в оперативному управлінні, дозволяє отримувати всі види статистичних даних, необхідних для фінансового і економічного аналізу, а також організувати спільну роботу всіх служб і тим самим скоротити як тимчасові, так і фінансові втрати, пов'язані з помилками персоналу або пацієнтів, має зворотний зв'язок. У реєстратурах поліклініки встановлюються програми, які дозволяють налагодити ведення електронних розкладів лікарів, планування прийому із записом пацієнтів (очний і по телефону), організовується одночасний доступ всіх реєстраторів в єдиний банк вільних талонів до лікарів, що дозволяє збільшити потік пацієнтів, не створюючи при цьому черг.

На наступному етапі налагоджується комп'ютерний облік усіх наданих пацієнту послуг - в стаціонарі, поліклініці, лабораторії, що дозволяє сформувати медичну статистику. В результаті цього кожен пацієнт отримує електронну історію хвороби, що містить всі відомості про звернення, випадках госпіталізації, надані

послуги та встановлені діагнози. В результаті можна буде отримувати весь комплекс медико-статистичної інформації як стандартизованої, так і по довільним параметрам.

На останньому етапі реєстраційно-статистична підсистема підтримує основний технологічний цикл роботи медичного закладу.

Лабораторна підсистема - лабораторна інформаційна система (ЛІС). У функції ЛІС входить управління всіма даними, які надходять з різних джерел (аналізатори, проведені вручну вимірювання, паперові документи) і об'єднання цих даних в єдину інформаційну базу даних клініко-діагностичної лабораторії. Підсистема забезпечує з'єднання практично з будь-якими автоматичними аналізаторами і дозволяє обмінюватися інформацією в режимі реального часу з будь-яким АРМ, що дає миттєвий доступ до готових результатів. ЛИС дозволяє скоротити непродуктивні витрати лабораторії, позбутися від рукописних журналів, відстежувати і оцінювати якість досліджень і отримувати всю необхідну звітну документацію.

2. Медична підсистема (електронна історія хвороби) - дозволяє лікарям в зручній формі зберігати і передавати один одному матеріали, пов'язані з діагностикою та лікуванням пацієнта, допомагає в науковій роботі, створює основу для віддалених лікарських консультацій по телемедичним каналах.

4. Формалізована електронна історія хвороби являє собою спеціалізовану базу даних, яка міститиме відомості про пацієнта і повний (в юридичному і медичному аспектах) набір документів про хід лікувально-діагностичного процесу, в тому числі:

- паспортні дані хворого;
- сигнальну інформацію (непереносимість лікарських засобів, спадкові захворювання і т.д.);
- анамнестичні дані;
- заключні діагнози, перенесені операції;
- результати лабораторних досліджень;

- результати інструментальних досліджень;
- диспансеризацію;
- анкету автоінтерв`юірування (скринінг) і ін.

5. Довідкова підсистема, надає лікарям довідкову інформацію щодо ведення лікувальної діяльності (наприклад, довідники лікарських засобів, лікарські новинки, посилання на медичні сайти і т.п.) Довідкова підсистема незалежна від перших трьох підсистем і може використовуватися в будь-який момент.

6. Аналітична підсистема інтегровану медичну і фінансову інформацію у вигляді, зручному для аналізу. Таким чином, вона допомагає керівництву виробляти обґрунтовані довгострокові рішення і контролювати ефективність перетворень, які проводяться в лікувальному закладі, удосконалювати стратегічне планування. Основою для роботи аналітичної інформаційної системи лікувального закладу є детальні персоніфіковані дані про склад і реальної вартості лікування кожного пацієнта. Ці дані підсумовуються по кожному підрозділу, лікаря, кабінету, за профілями лікування, по компаніям страховок і джерела фінансування.

1.2.4. Рівні медичних інформаційних систем

Прийнято виділяти п'ять основних рівнів автоматизації медичних інформаційних систем. На першому рівні знаходяться медичні записи, модернізовані за допомогою автоматизації. Даний рівень описується тим, що всього близько половини інформації про пацієнта заноситься в інформаційну систему, і далі може бути видана лікуючого лікаря в формі різних звітів і бланків. Дану інформаційну систему можна назвати свого роду автоматизованим оточенням паперової технології прийому пацієнта. Ці автоматизовані системи найчастіше поширюються на реєстрацію пацієнта, виписному листи, трансформації всередині лікарні,

ведення даних про діагностику, призначення лікарів, підсумки операцій, і так далі.

Другий рівень медичних інформаційних систем вдає із себе систему автоматизованого запису. На даному рівні відбувається індексація, сканування, і запам'ятовування в системі, де зберігаються зображення в електронному вигляді тих документів, які не були внесені в базу даних раніше (маються на увазі в основному дані, отримані за допомогою пристроїв діагностики, одержуваних у формі різного роду бланків і т .п.).

На третьому рівні розвитку МІС починається впровадження так званих електронних медичних записів. Для застосування даної технології необхідно, щоб інфраструктура медичної установи була розвинена належним чином, щоб вона могла надати можливості зберігання і обробки даних безпосередньо з робочого місця лікаря. Вони повинні проходити ідентифікацію за допомогою системи, де їм будуть видані певні права доступу. Передбачається, що на цьому рівні розвитку системи електронний запис може мати більш важливе значення в діяльності прийняття рішень і кооперації з експертними системами, при постановці лікарем діагнозу, вказівки необхідних призначень і так далі.

Четвертий рівень медичних інформаційних систем має назву система електронних медичних записів, де дані про пацієнта можуть мати набагато більше інформаційних джерел. Тут міститься все основні дані про кожного пацієнта, які можуть мати різні джерела, тобто різних медичних установ. Даний рівень розвитку системи вимагає наявність багатонаціональної або всеросійської системи для ідентифікації пацієнтів, спеціальна структура інформації, загальна система термінів, кодування та ін.

П'ятий рівень розвитку МІС несе в собі назву електронний запис про здоров'я. Даний рівень відрізняється від попереднього тим, що тут сформовано існування практично необмеженої кількості джерел даних про пацієнта. Має місце поява різного роду

даних з інших областей медицини, поведінкової діяльності і так далі.

1.2.5 Технології для розробки МІС

Медичні інформаційні системи повинні підтримувати web-інтерфейс. Тому сучасну МІС ефективніше проєктувати як web-додаток. Для такого роду завдань широко застосовується архітектура MVC (Module-View-Controller). На сьогоднішній день є безліч платформ реалізують даний підхід. Найбільш відомі з них:

- Ruby on Rails;
- Play Framework;
- Apache Tapestry;
- ASP.NET MVC;
- Spring Framework MVC;
- GrailsBci.

Вони надають більш менш схожий функціонал для розробки повноцінного MVC-додатки. Відрізняються вони в основному мовами програмування: для Ruby on Rail - це Ruby, для ASP.NET MVC - C #, SpringFramework MVC і Apache Tapestry - Java, Play Framework - Scala і Java, Grails - Groovy. Виходячи з цього, можна узагальнити технологічні аспекти вищевказаних платформ, які необхідні для розробки сучасної МІС:

– шаблони web-сторінок на стороні сервера. Більшість сучасних платформ підтримують використання шаблонів для розробки користувальницького інтерфейсу. Основна ідея полягає в їх перевикористанні для швидкого створення нових елементів на рівні View (подання до моделі MVC) і динамічної генерації web-сторінок. Для платформ підтримують стек технологій Java EE- це java server pages, для ASP.NET active server pages, для Grails - groovy server pages, для Play Framework - це scala шаблони. Відповідно всі вищевказані платформи мають такий функціонал в тому чи іншому вигляді: присутні лише відмінності в синтаксисі;

– інтеграція з сучасні СУБД. Можливість роботи з реляційними базами даних (PostgreSQL, MS SQL, MySQL, H2SQL і ін.), а також з NoSQL рішеннями (MongoDB, OrientDB і ін.);

– інтеграція з хмарними сервісами. Можливість розмістити розроблений додаток в хмарі, скориставшись сервісами PaaS. Прикладами можуть бути Jelastic і Heroku - для java-додатків, або Microsoft Azure - для ASP.NET;

– кешування. Для прискорення роботи web-додатки можна зберігати результати роботи як на стороні клієнта, так і на серверної. сучасні MVC-платформи підтримують такий механізм. Скажімо, для ASP.NET – це реалізується за допомогою атрибутів, які визначають параметри зберігання кешу, а для Play Framework - це реалізується через java-анотації, та доступ до кешованих даних можливий безпосередньо через API платформи;

– базові елементи безпеки для web-додатки на рівні платформи: екранування спеціальних символів при роботі з web-сторінками, асоціювання ключа безпеки з сесіями користувачів для запобігання несанкціонованому доступу.

– можливість конфігурації маршрутизації запитів на web-сервер з розділенням на POST і GET. Наприклад, в Play Framework для цього окремо налаштовується конфігураційний файл, де явно вказується шлях, вид запиту і використовуваний контролер.

1.3 Керування ризиками в медичних системах

Медичні інформаційні системи, що призначені для автоматизації медичних процесів та є їх складовими, можуть створювати додаткові фактори ризику для ресурсів цього процесу та повинні бути враховані при організації ризик-менеджменту конкретного медичного процесу.

Управління ризиком (risk management) – це систематичне застосування політики, процедур і практичних методів управління для задач аналізування, оцінювання, контролювання і моніторингу ризику [10].

Згідно [11] до медичних виробів серед іншого відносять також програмне забезпечення, що застосовується як окремо, так і в поєднанні між собою (включаючи програмне забезпечення, передбачене виробником для застосування спеціально для діагностичних та/або терапевтичних цілей та необхідне для належного функціонування медичного виробу). Тому до медичних інформаційних систем діють ті ж самі вимоги, що і для медичних виробів стосовно забезпечення якості та безпеки функціонування, зокрема той її частини, яка стосується аналізу та оцінки ризиків.

Зокрема, безпечним вважається виріб, що використовується згідно передбаченого застосування та для якого відсутній неприпустимий ризик.

Ризик – поєднання вірогідності виникнення шкоди і ступеня тяжкості цієї шкоди [12].

Використання конкретних процедур при управлінні ризиками можуть відрізнятися, але незмінним залишається необхідність ідентифікації небезпек, оцінки ризику та розробки, якщо це необхідно рекомендацій по зниженню ризику.

1.3.1 Етапи процесу управління ризиком

Як приклад на рисунку 1.7 наведено схематичне зображення процесу управління ризиком для медичних виробів [10].

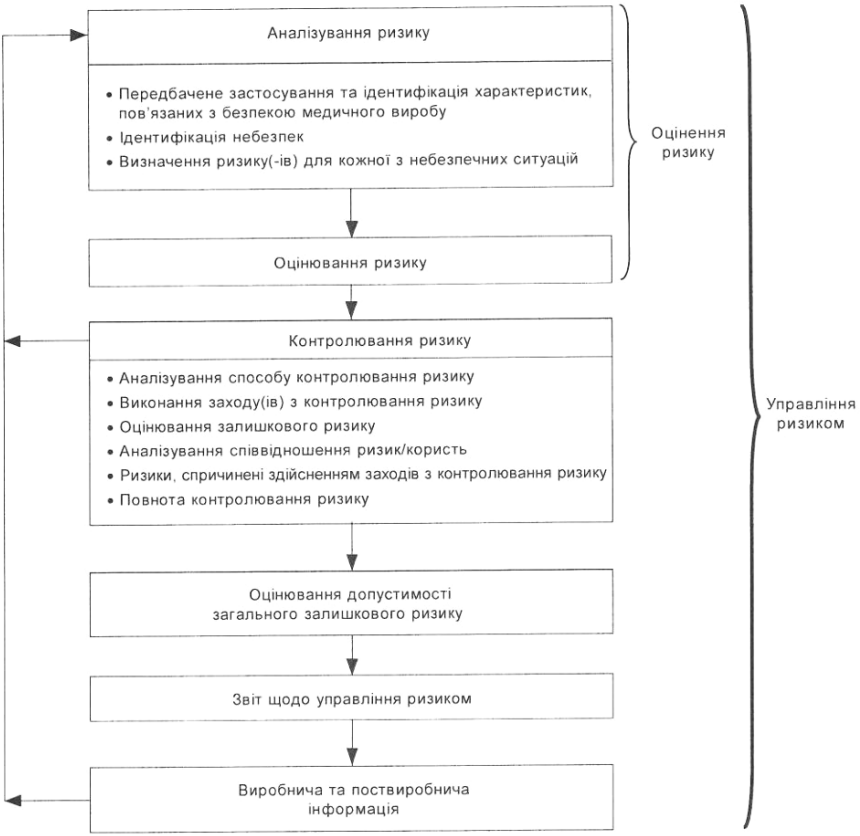


Рисунок 1.7 – Схематичне зображення процесу управління ризиком

До початку проведення дій з управління ризиком необхідно чітко окреслити галузь застосування: визначити об'єкт оцінки ризиків та активи, що представляють цінність.

Стосовно медичної сфери об'єктом оцінки ризику може бути як окремий медичний виріб, так і процеси, пов'язані з ним.

При визначенні активів слід враховувати, що основу будь-якого об'єкту чи процесу складають виконувані функції, для здійснення яких потрібні ресурси:

– людські – учасники процесу;

– виробничі;
– матеріальні (в тому числі навколишнє середовище);
– інформаційні;
– інтелектуальні – знання і повноваження учасників.

Управління ризиками для медичних виробів та процесів перш за все має першорядне значення для людських ресурсів: пацієнтів, операторів та інших осіб, що беруть участь в медичному процесі, та для навколишнього середовища.

Передбачене застосування та ідентифікація характеристик, пов'язаних з безпекою

Після того, як була окреслена галузь застосування, проводять роботи з аналізування ризику, за якими встановлюють межі передбаченого застосування та ідентифікацію характеристик, пов'язаних з безпекою.

Згідно вимог [10] виробник має задокументувати призначене застосування медичного виробу та передбачуване неправильне застосування. Також виробник визначає характеристики (якісні та кількісні), які можуть впливати на безпеку, та, за можливістю, рівень їх допустимого значення.

В результаті отриманні матеріали стають базисом для ідентифікації небезпек.

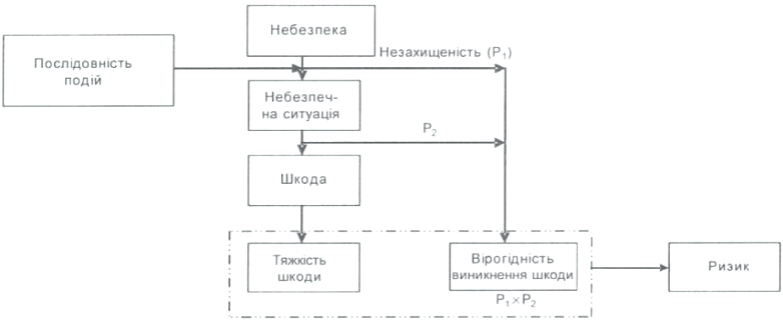
Шкода – фізична травма або збиток здоров'ю, майну або довкіллю [12].

Небезпека – потенційне джерело шкоди [12].

Ідентифікація небезпек.

Небезпека може бути створена завдяки небезпечній ситуації за умови деяких обставин. Тому треба звертати увагу на послідовності або комбінації подій, що можуть викликати небезпечну ситуацію як за умови нормального стану виробу, так і у стані збою.

На рисунку 1.8 наведено взаємозв'язок між небезпекою, послідовністю подій, небезпечною ситуацією та шкодою [10].



Познаки:
P_1 – вірогідність виникнення небезпечної ситуації;
P_2 – вірогідність того, що небезпечна ситуація спричинить шкоду.

Рисунок 1.8 – Взаємозв'язок між небезпекою, послідовністю подій, небезпечною ситуацією та шкодою

Слід зауважити, що збій в роботі не завжди є гарантом небезпечної ситуації з заподіянням шкоди.

Розрізняють два види збоїв: випадкові та систематичні.

Як приклад випадкового збою можна навести відмову в роботі апаратного забезпечення у довільний момент часу, що призводить до зниження або втрати здатності виконувати закладені функції.

Причинами систематичних збоїв стають зазвичай помилки у будь-якому виді діяльності. Такі помилки можуть виникати як у апаратному, так і у програмному забезпеченні.

Приклад систематичного збою: при розробці баз даних програмного забезпечення не було передбачено випадку надмірного заповнення даними, що може призвести до їх втрати та, як наслідок, некоректного функціонування системи в цілому.

На цьому етапі ідентифікації небезпек також слід звернути увагу на передбачуваних користувачів медичного виробу (або учасників медичного процесу), бо їх кваліфікація може суттєво впливати на виникнення потенційних небезпек. Наприклад, медичний виріб призначений для кваліфікованих медичних працівників чи його буде використовувати звичайний споживач.

Для проведення більш якісного аналізу та отримання більш достовірних даних в майбутньому доцільно проводити акумулювання інформації щодо відомих та передбачуваних небезпек, пов'язаних з медичним виробом або процесом як за умов нормальної експлуатації, так і при аварійних ситуаціях.

Корисним буде зробити огляд досвіду користування такими самими чи подібними виробами, що буде містити перелік типових небезпечних ситуацій. При цьому слід враховувати власний досвід, а також досвід інших виробників, доступний з різних джерел інформації.

Визначення ризиків

Це останній етап з аналізування ризику. Для кожної небезпечної ситуації, що була ідентифікована, слід визначити ризик, до якого така ситуація може призвести.

Складові частини ризику, вірогідність та тяжкість наслідків, зазвичай розглядаються окремо.

Інформація, що була отримана при використанні наступних підходів, є вирішальною при визначенні вірогідності:

- використання відповідних історичних даних;
- передбачення вірогідності за допомогою аналітичних та симуляційних методів;
- використання експериментальних даних;
- визначення надійності;
- виробничі дані;
- пост виробнича інформація;
- використання оцінки експертів [10].

На практиці доцільно використання дискретної кількості рівнів вірогідності, що значно спрощує процес аналізування. Особливо ефективно присвоєння кількісних значень дискретним рівням.

Але існує певна категорія ризиків вірогідність яких визначити неможливо або надзвичайно важко. До таких ризиків, наприклад,

відносяться ризики, що були викликані систематичними збоями: помилки в роботі програмного забезпечення; відмова або навмисне псування медичного виробу; нові небезпеки, що є не до кінця зрозумілими тощо.

В таких випадках встановлюють широкий діапазон для вірогідності або припускають, що вона не вища певних показників.

Для того щоб ідентифікувати тяжкість потенційної шкоди на практиці також доцільно використання дискретних кількісних рівнів тяжкості, що значно полегшує цей процес.

Приймається рішення щодо необхідної кількості якісних категорій та їх визначення. Вони можуть бути представлені лінгвістично (наприклад, не потребує медичного втручання, потребує медичного втручання, потребує госпіталізації, призводить до летальних наслідків та ін.) або символічно (L1, L2 тощо). При символічному визначенні кожен символ має бути чітко визначений.

При будь-якому представленні кожна з категорій тяжкості потенційної шкоди не повинна містити елементів ймовірності.

Хоча кількісна оцінка ризику і більш прийнятна, але за умови неможливості її проведення можуть бути достатніми якісні методи визначення ризику.

Коли вірогідність та ступінь тяжкості нанесення шкоди неможливо розрахувати за відсутності достовірної кількісної інформації використовують інші способи визначення ризику та візуалізації значення цього визначення.

Це може бути, наприклад, двомірний графік ризику (рисунок 1.9), де по кожній небезпечній ситуації визначення тяжкості та вірогідності виникнення шкоди можна представити у вигляді окремих точок на площині або за допомогою матриці ризиків.

Двомірний графік ризику – це зображення, що має орієнтовне значення компонентів ризику відносно один до одного та полегшує процес прийняття рішення щодо контролювання ризику.

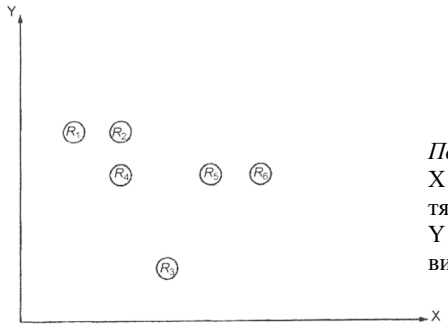


Рисунок 1.9 – Приклад графіку ризиків

Також типовим підходом якісного аналізування ризиків є використання матриці для опису вірогідності та тяжкості ризиків у кожній небезпечній ситуації.

Чітко визначають кількість рівнів як вірогідності, так і тяжкості та їх наповнення. В таблиці 1.1 представлені три прості приклади якісних рівнів тяжкості, а в таблиці 1.2 – три спрощені якісні рівні вірогідності [10].

Таблиця 1.1 – Приклади якісних рівнів тяжкості

Рівні тяжкості	Можливий опис
Значний	Летальний випадок або зміна анатомічної будови тіла
Помірний	Зворотні чи незначні ушкодження
Незначний	Не призводить до ушкоджень або призводить до незначних ушкоджень

Таблиця 1.2 – Спрощені приклади якісних рівнів вірогідності

Рівні вірогідності	Можливий опис
Високий	Можливий, трапляється часто
Середній	Можливий, але не часто трапляється
Низький	Навряд трапляється, рідко

Далі формується матриця ризиків, де в найменуваннях рядків представлені лінгвістичні значення рівнів вірогідності, а в найменуваннях стовпців – рівні тяжкості. До кожної комірки матриці вносять відповідні очікувані ризики. Приклад матриці ризиків наведено у таблиці 1.3.

Таблиця 1.3 – Приклад матриці ризиків

		Якісні рівні тяжкості		
		Значний	Помірний	Незначний
Якісні рівні вірогідності	Високий		R_2	R_1
	Середній	R_3	R_4	
	Низький			R_5, R_6

Інколи при встановленні рівнів вірогідності використовують напівкількісні шкали, де кожному рівню вірогідності ставлять у відповідність деякий діапазон значень. Приклад напівкількісних рівнів вірогідності наведено у таблиці 1.4.

Таблиця 1.4 – Приклади напівкількісних рівнів вірогідності

Рівні вірогідності	Приклади діапазону вірогідності
Часто	$\geq 10^{-3}$
Ймовірно	$< 10^{-3}$ та $\geq 10^{-4}$
Час від часу	$< 10^{-4}$ та $\geq 10^{-5}$
Рідко	$< 10^{-5}$ та $\geq 10^{-6}$
Малоймовірно	$< 10^{-6}$

Для визначення тяжкості нанесення шкоди вкрай рідко використовують кількісні або напівкількісні шкали через важкість порівняння в одних одиницях, наприклад, кількісного показника смерті та кількісного показника терміну непрацездатності. Виняток становить, наприклад, ситуація, коли аналізування

стосується тільки ризику нанесення матеріальної шкоди, яка може бути визначена в грошових одиницях.

Кількість якісних рівні як для вірогідності, так і для тяжкості наслідків визначає виробник, виходячи з принципів оптимальності та необхідної точності оцінок.

Слід враховувати, що матриці з трьома якісними рівнями не завжди можуть бути достатньо точними для прийняття рішення. Більше рівнів – більша точність. Але при цьому матриці, що містять більше ніж п’ять рівнів потребують значно більше інформації для присвоєння ризику до того чи іншого рівня.

Обґрунтування щодо розрядності матриці ризиків, що складається з кількості рівнів вірогідності та кількості рівнів тяжкості наслідків, повинно бути чітко задокументовано виробником.

Слід зауважити, що виробник для кожної групи однотипних виробів може встановлювати свої рівні визначень та матриці ризиків. Тому що стосовно сфери застосування виробу можуть бути використані різні заходи вірогідності. Наприклад, «вірогідність нанесення шкоди за одно застосування», «вірогідність нанесення шкоди за одну годину роботи», «вірогідність нанесення шкоди на один виріб» тощо.

Оцінювання ризику

Для кожного визначеного ризики приймається рішення щодо його допустимості та необхідності його зменшення. Роблять це за допомогою критеріїв допустимості ризиків. Виробник може класифікувати ризики за ступенем допустимості для визначення тих з них, які необхідно зменшити.

Для визначення допустимості ризиків виробник може використовувати відповідні стандарти на конкретний тип виробу (за їх наявності), порівняння рівнів ризику виробів, що знаходяться в експлуатації, дані клінічних досліджень, наявну достовірну інформацію про досвід інших виробників з доступних джерел.

Застосування критеріїв допустимості може бути відображено у матриці ризиків. Наприклад за допомогою кольору, як показано на рисунку 1.10.

		Якісні рівні тяжкості				
		Катастрофічний	Критичний	Серьозний	Другорядний	Незначний
Якісні рівні вірогідності	Часто					
	Ймовірно		R_2			R_6
	Час від часу		R_3		R_5	
	Рідко	R_1				
	Малоймовірно			R_4		

Познаки:

– недопустимий ризик;
– допустимий ризик

Рисунок 1.10 – Приклад матриці оцінювання ризиків з колірним показником критерію допустимості

Слід зазначити, що зона матриці може бути поділена на більше ніж дві зони критерію допустимості. Зробити це можна за допомогою класифікації кожного визначеного ризику до певної категорії або класу.

Для цього необхідно визначити та описати необхідну кількість категорій ризику – ранги. У таблиці 1.5 наведено приклад лінгвістичного опису рангів ризику.

Таблиця 1.5 – Ранги ризиків

Ранги ризиків	Опис рангів ризиків
I	Недопустимий ризик
II	Небажаний ризик, який може бути допустимим тільки, якщо зниження ризику непрактично й веде до непропорційного збільшення витрат у порівнянні з досягнутим поліпшенням
III	Допустимий ризик, якщо витрати на зниження ризику не перевищують досягнутих поліпшень
IV	Нехтовно малий ризик

Тоді матриця ризиків, поділена на чотири зони за критеріями допустимості, може мати вигляд як на рисунку 1.11.

		Якісні рівні тяжкості				
		Катастрофічний	Критичний	Серьозний	Другорядний	Незначний
Якісні рівні вірогідності	Часто					
	Ймовірно		R_2			R_6
	Час від часу		R_3		R_5	
	Рідко	R_1				
	Малоймовірно			R_4		

Познаки:

– I ранг; – III ранг;

– II ранг; – IV ранг

Рисунок 1.11 – Приклад матриці оцінювання ризиків, що поділена на чотири зони

Контролювання ризику

На наступному етапі визначають заходи, необхідні для зменшення ризиків до прийнятного рівня. Такі заходи ще називають контрзаходами. Вони здатні зменшувати тяжкість можливої шкоди або її вірогідність, або обидві ці складові ризику разом. Класифікація контрзаходів та їх співвідношення наведені нижче.

Стандартними способами зі зменшення ризику згідно [13] є наступні три:

– безпека передбачена конструкцією;

– запобіжні заходи у самому медичному виробі або виробничому процесі;

– інформування щодо безпеки.

Саме така послідовність використання наведених способові вкрай важлива. Тобто медичний виріб апріорі необхідно проєктувати з максимальною безпечністю для користувача. При цьому можуть бути усунені певні небезпечні фактори, зменшені вірогідність нанесення шкоди або її ступінь тяжкості. Якщо це неможливо, то необхідно передбачити запобіжні заходи, такі як візуальні та звукові сигнали тривоги або автоматичні вимикачі тощо. І останній за перевагою спосіб зменшення ризиків – це письмові заходи: попередження або протипоказання.

Після визначення всіх необхідних заходів щодо зниження підсумкового ризику їх впроваджують до виробу або процесу, що повинно бути обов'язково перевірено.

Перевіряють за допомогою контролю впровадження заходів у закінченому проєкті або фактичним підтвердженням ефективності впроваджених заходів на зменшення ризику, для чого можуть знадобитися додаткові підтверджувальні дослідження.

Слід враховувати, що деякі з визначених та впроваджених заходів зниження ризиків можуть внести додаткові умови для створення небезпечної ситуації з заподіянням шкоди. Такими новими або збільшеними ризиками, що є наслідком вжитих заходів по контролюванню ризиків, необхідно управляти так само як і будь якими іншими, відповідно до зазначених вище методів та підходів.

Після впровадження заходів зі зниження ризику необхідно оцінити залишковий ризик, використовуючи ті ж самі методи та критерії допустимості ризиків, які були використані на етапі аналізування ризиків.

Якщо ж залишковий ризик після повторного оцінювання попадає в зону допустимого ризику, то виробник має визначити які саме з залишкових ризиків необхідно зазначити, яку інформацію, в якій формі та наскільки детально необхідно викласти в супровідних документах. Робиться це задля пояснення

залишкового ризику так, щоб користувачі активно вживали відповідних заходів для мінімізації впливу залишкового ризику.

Якщо ж залишковий ризик все ж таки залишається в зоні неприпустимих ризиків слід застосувати додаткові заходи зі зниження ризику. Ця багаторазова процедура повторюється доки залишковий ризик не буде в зоні допустимого.

Але ймовірна ситуація, коли практичний спосіб зменшення ризику до допустимого рівня відсутній або є недоцільним, тоді виконують аналізування співвідношення користь/ризик, для визначення переваги медичної користі застосування виробу над залишковим ризиком. Для цього застосовують всі зібрані і перевірені дані та наявну літературу.

Якщо докази свідчать, що користь використання медичного виробу не перевищує залишковий ризик, то такий ризик вважають неприпустимим і від такого проєкту треба відмовитися.

Однак іноді значні ризики можна вважати допустимими, якщо медичний виріб приносить більше користі ніж шкоди. В такому випадку виробник документально обґрунтовує залишковий ризик з зазначенням заходів безпеки по використанню виробу.

Користь встановлюють за покращенням стану здоров'я згідно оцінок наступних факторів:

– ефективність використання;

– очікуваний клінічний результат;

– дані щодо користі інших способів лікування.

Надійність доказів гарантує впевненість у проведеній оцінці. При цьому слід враховувати, що є проблема з порівнянням різних наслідків. Наприклад, складно порівнювати біль з втратою рухливості. Це певною мірою зумовлено відмінністю початкової проблеми від побічних дій.

Також важко враховуються нестабільні наслідки, що можуть бути результатом реабілітаційного періоду або довготривалих ефектів [10].

Оцінка співвідношення ризик/користь є специфічною для кожного виробу, тому єдиного уніфікованого підходу не існує. При такому виді аналізування необхідно враховувати технічний, клінічний, регуляторний, економічний, соціологічний та політичний контекст.

Безпосереднє порівняння користі та ризику можна проводити за умови використання спільної шкали, що дозволяє отримати кількісні оцінки. За непрямих порівнянь використовують якісні методи оцінювання.

При необхідності проведення клінічних досліджень для встановлення допустимості співвідношення користі та залишкового ризику вони регулюються [14–15], що дозволяє отримати кількісну оцінку ризиків та користі.

Оцінювання допустимості загального залишкового ризику

Після проведення всіх попередніх етапів виробник повинен поглянути на процес управління ризиками з точки зору широкої перспективи і оцінити сукупний вплив залишкових ризиків та прийняти рішення щодо подальшої доцільності роботи над виробом. Ймовірна ситуація коли загальний залишковий ризик перевищує встановленні критерії допустимості, тоді як окремі залишкові ризики не перевищували таких критеріїв. Особливо це стосується складних систем з великою кількістю можливих ризиків та їх комбінацій.

Такий аналіз повинні здійснювати спеціалісти, що є експертами в конкретній предметній області та мають відповідний досвід та повноваження.

Рішення щодо доцільності враховує, що всі можливі ризики повинні бути знижені до найнижчого можливого рівня за умови практичної здійсненності.

Практична здійсненність складається з технічної здійсненності та економічної доцільності.

Технічна здійсненність – це здатність виробника знизити ризик незалежно від фінансових витрат.

Економічна доцільність передбачає зниження ризику за умови, що медичний виріб залишається економічно вигідним проєктом.

Звіт щодо управління ризиком та виробнича і пост виробнича інформація

В результаті всіх проведених робот формують звіт щодо управління ризиками, де коротко викладають остаточні результати процесу управління ризиком.

Однак на цьому робота не закінчується. Аналіз і оцінка ризиків використовується для забезпечення безперервного управління ризиками.

Ризиками, які стали наявними на одному з етапів медичного процесу, можна управляти за допомогою відповідних заходів абсолютно на іншому етапі. Тому принципи управління ризиком повинні бути застосовані з моменту зародження медичного процесу і впродовж всього його існування.

Виробники повинні здійснювати моніторинг виробничої і пост виробничої інформації для отримання даних, що можуть впливати на визначення ризиків, а отже і можуть бути причиною перегляду звіту та покращення процесу управління ризиками. Використання пост виробничої інформації дозволяє зробити процес управління ризиком повторюваним зі зворотнім зв'язком.

1.3.2 Методи оцінки та управління ризиками

Пріоритетних методів для оцінки ризиків не існує. Є загальні рекомендації по доцільності застосування конкретних методів на різних етапах аналізу та оцінки ризику. Відповідальним за вибір відповідного методу є виробник.

Аналіз ризиків можна умовно розділити на два взаємодоповнюючі один одного види: якісний і кількісний. Якісний аналіз має на меті ідентифікувати фактори, області й види ризиків. Кількісний аналіз ризиків повинен дати можливість

чисельно визначити розміри окремих ризиків та ризику об'єкта в цілому.

Якісний аналіз ризиків

Вибір якісного методу аналізу небезпек залежить від мети аналізу, призначення об'єкта, його стану й складності.

Якісні методи аналізу містять наступні:

- попередній аналіз небезпек (Preliminary Hazard Analysis – PHA) [16];
- метод аналізу небезпеки та працездатності (Hazard and Operability Study – HAZOP) [17];
- методи перевірочного листа (Check-list) та «Що буде, якщо ...?» («What – If») [18];
- аналіз видів та наслідків відмов (Failure Mode and Effects Analysis – FMEA) [19];
- аналіз видів наслідків та критичності відмов (Failure Mode, Effects and Critical Analysis – FMECA) [19];
- графічно-аналітичні методи аналізу небезпек за допомогою дерев: дерева відмов (Fault Tree Analysis – FTA); дерева подій (Event Tree Analysis – ETA); дерева рішень, причин, наслідків тощо [18, 20];
- система аналізування ризиків та критичних контрольних точок (Hazard Analysis and Critical Control Points – HACCP) [21];
- методи аналізування суперечливих вимог, попереджень, інструкцій з використання тощо [10];
- логіко-ймовірнісне моделювання [22];
- аналіз небезпек методом потенційних відхилень [18];
- аналіз помилок персоналу [18];
- причино-наслідковий аналіз [18] та інші.

Зазвичай вказані методи не використовуються по одинці. Для досягнення найбільшої ефективності якісного аналізу ризиків проводять покрокове аналізування послідовності подій з застосуванням комбінації декількох методів.

Для якісного аналізу ризиків спочатку доцільно застосувати Попередній аналіз небезпек, який містить не тільки попереднє виявлення елементів системи або подій, які ведуть до небезпечних ситуацій, а також коригувальних заходів (контрзаходів) для усунення небезпек. Результатом Попереднього аналізу небезпек будуть: перелік небезпек, потенційно небезпечне місце або елемент системи й коригувальні впливи. Таким чином, даний метод дозволяє виділити з великого числа можливих потенційно небезпечних подій пріоритети й види небезпек, які слід розглядати більш докладно.

На основі попереднього аналізу небезпек надалі можна застосувати різні графічно-аналітичні методи аналізу небезпек за допомогою дерев або детальний кількісний аналіз.

Кількісний аналіз ризиків

При кількісному аналізі під терміном “ризик” розуміють багатокомпонентну величину, яка характеризується нанесенням шкоди від впливу того або іншого небезпечного фактору, імовірністю виникнення розглянутого фактору та невизначеністю у величинах як шкоди, так і ймовірності.

Залежно від мети аналізу для чисельної оцінки ризику використовують різні математичні формулювання [18].

Згідно класичному визначенню ризик характеризується двома величинами – імовірністю події P і наслідками U (наприклад, у вигляді шкоди):

$$R = P \cdot U$$

За необхідності можна використовувати визначення ризику як імовірності перевищення меж x:

$$R = P\{\xi > \mathrm{x}\},$$

де ξ – випадкова величина.

Якщо кожній небажаній події, що відбувається з імовірністю Pi, відповідає збиток Ui, то величина ризику буде являти собою очікувану величину збитку U*:

$$R = U^* = \sum_{i=1}^{n} P_i \cdot U_i$$

Якщо всі ймовірності настання небажаної події однакові (Pi = P, i=1 … n), то слід:

$$R = P \cdot \sum_{i=1}^{n} \mathrm{U_i}$$

При оцінці ризиків медичних виробів одночасно існує небезпека як здоров'ю, так і матеріальним цінностям або довкіллю. В таких випадках ризик доцільно представляти у векторному вигляді з різними одиницями виміру за координатними осями:

$$\vec{R} = \vec{U} \cdot \vec{P}.$$

Перемножування в правій частині цього рівняння проводиться покомпонентно, що дозволяє порівнювати ризики різного походження.

Індивідуальний ризик можна визначити як очікуване значення завданої шкоди U* за інтервал часу T та віднесене до групи людей чисельністю M осіб:

$$R = U^*/(M \cdot T).$$

Загальний ризик для групи людей (колективний ризик) розраховують:

$$R = \mathrm{U^*/T}.$$

Проте необхідно враховувати, що здійснення кількісної оцінки зустрічає і найбільших труднощів, які пов'язані з тим, що для цього виду аналізу необхідна відповідна вхідна достовірна інформація. А для рідкісних подій та для нових розробок, які за своєю суттю є унікальними виробами з новими функціональними властивостями і взаємозв'язками між елементами конструкції, низкою нововведень, що до цього не використовувалися, з одного боку немає достатньої статистики, щоб спиратися на досвід, а з іншого – немає теорії, яка дозволяла б виводити ці величини, виходячи з первинних принципів.

В цьому випадку на ряду з клінічними дослідженнями можливо застосування експертного підходу: ймовірності різноманітних подій, зв'язки між ними і наслідки заподіяної шкоди визначають не обчисленнями, а опитуванням досвідчених експертів.

1.3.3 Методи експертного оцінювання

Методи експертних оцінок – це методи організації роботи з фахівцями, експертами, і обробки думок експертів.

Основним завданням експертного оцінювання є:

– забезпечення адекватної оцінки об'єкта експертизи;

– виділення альтернативних варіантів рішень, що можуть бути реалізовані та здатні привести до мети;

– обґрунтований вибір самої прийнятної з альтернатив[23].

Колективне експертне оцінювання дозволяє знизити суб'єктивність індивідуальних думок експертів і підвищити ефективність прийняття колективного рішення.

У загальному випадку процедура підготовки та прийняття колективного рішення може бути розбита на наступні етапи: попередній аналіз проблеми та постанова задачі, розробка регламенту проведення експертизи, формування експертної групи,

отримання індивідуальних експертних оцінок, перевірка узгодженості думок експертів і формування колективної оцінки, прийняття колективного рішення.

Класифікація методів експертного оцінювання

На кожному з цих етапів застосовуються різні методи, які умовно можна розбити на наступні групи:

– методи колективного обговорення, що використовуються для попереднього аналізу проблем (інтерв'ю, анкетування, метод "мозкового штурму", метод Дельфі тощо);

– методи формування груп експертів і визначення компетентності експертів (метод самооцінки, метод взаємооцінки, метод оцінки за формальними показниками анкети, тестування, теорія прецедентів, методи, засновані на результатах колишньої діяльності експертів тощо);

– методи оцінки узгодженості думок експертів (методи знаходження коефіцієнтів конкордації, коефіцієнт рангової кореляції Спірмена і Кендалла тощо);

– методи формування колективної експертної оцінки (методи обчислення середніх (середньоарифметичне, середньогеометричне, медіана, мода тощо), метод медіани Кемені, обчислення середнього рангу та ін.);

– методи оцінювання альтернатив (метод аналізу ієрархій, методи ранжування альтернатив та ін.) [24].

У методах оцінювання альтернатив широко поширені такі види експертних суджень:

– парні порівняння. З наявної сукупності об'єктів експерту по черзі пред'являються можливі пари об'єктів. Для кожної пари експерт вказує, який з двох об'єктів краще іншого за певною ознакою;

– множинні порівняння. Із сукупності об'єктів складаються не пари, а набори об'єктів. Експерт впорядковує об'єкти в наборі в порядку переваги;

– ранжування. Експерт впорядковує наявні об'єкти за зменшенням (або за зростанням) з точки зору заданої ознаки;

– класифікація. Експерт розділяє всю сукупність об'єктів на кілька (можливо, заздалегідь заданих) класів;

– безпосередня оцінка. Експерт приписує наявним об'єктам числові значення в шкалі значень даної ознаки [23].

Для конкретного завдання експертизи можуть використовуватися різні види суджень або їх поєднання.

Розглянемо постановку задачі експертного оцінювання в загальному вигляді. Нехай існує множина об'єктів оцінювання X={xi}, i=1,2,…,n. Тоді Ω – множина припустимих оцінок цих об'єктів, а C(Ω) – функція вибору, яка зіставляє кожній підмножині $\Omega' \subseteq \Omega$ її частину. Кожний з p експертів оцінює n альтернатив як $a_{ij} = C_{ij}(\Omega) \in \Omega$, де i=1,2,…,n та j=1,2,…,p.

В результаті колективного експертного оцінювання формується матриця n×p $A^{coll} = \|a_{ij}\|$ оцінок i-го об'єкту j-м експертом.

При цьому в залежності від задачі, що вирішується, множина припустимих значень визначається як [23]:

– $\Omega = \{0, 1\}$, може бути визначено завданням попарного порівняння об'єктів a' та b'

$$C(\Omega) = \begin{cases} 1, & \textit{якщо } a' \textit{ краще } в', \\ 0, & \textit{якщо інакше.} \end{cases}$$

$\Omega = \{\langle 1,2,...,m\rangle, \langle 1,3,...,m,2\rangle, ..., \langle m, m-1,...,1\rangle\}$, яке складається з множини перестановок довжини m, що є характерним для вирішення задачі ранжування. Остання полягає в упорядкуванні об'єктів, що утворюють систему значень ознак. $C(\Omega) = \langle i_1, i_2, ..., i_m \rangle$, де i_j – номер j-го об'єкту при вказаному упорядкуванні.

$\Omega = \{1,\ldots,l\}$. Задача класифікації може полягати в співвідношенні $x \in S$ до однієї з l підмножин $S_1,\ldots,S_l$, де $S = S_1 \cup \ldots \cup S_l$, та при цьому $C(\Omega) = i$, якщо $x \in S_i$.

$\Omega = E_m$. Відповідне завдання чисельної оцінки може полягати у зіставленні об'єкту оцінювання одного або декількох чисел, причому $C(\Omega) = d$, якщо оцінкою об'єкта є вектор $d \in E_m$.

Оцінка узгодженості думок експертів

При оцінці об'єктів дослідження необхідна оцінка ступеня узгодженості думок експертів. Отримання кількісної міри узгодженості дозволяє більш обґрунтовано інтерпретувати причини розбіжності думок [25].

Для того щоб оцінити узгодженість думок декількох експертів при оцінюванні ряду об'єктів (альтернатив) можна скористатися коефіцієнтом конкордації (узгодженості) Кендалла і Сміта [16*], що широко застосовується при ранжуванні:

$$W = \frac{12 \cdot S_w}{p^2\left(n^3 - n\right)}, \qquad (1.1)$$

де $S_w = \sum_{j=1}^{n}\left(\sum_{i=1}^{p} a_{ij} - \frac{p(n+1)}{2}\right)^2$.

Тут a_{ij} – ранг і- го об'єкту, який присвоїв йому j-й експерт;

p – кількість експертів;

n – кількість об'єктів (альтернатив), що оцінюються.

При наявності зв'язних рангів, тобто присутності збігу місць, коефіцієнт конкордації (1.1) набуде вигляду:

$$W = \frac{12 \cdot S_w}{p^2\left(n^3 - n\right) - p\sum_{j=1}^{p} T_j}$$

де $T_j = \sum_{i=1}^{k} \left(t_i^3 - t_i\right)$;

k – кількість зв'язаних (об'єднаних) рангів;

t_i – кількість елементів у і-й зв'язці для j-го експерта.

Отримані величини порівнюють з критичними значеннями при заданій довірчій ймовірності (α = 0,95 и α = 0,99). Критичні значення SW(α) – довідкова величина [25].

При n > 10÷15 і відсутності кореляції величина p(n-1)·W розподілена приблизно як χ^2 зі f = n-1 ступенями свободи. Отже критичне значення дорівнює:

$$W_\alpha = \frac{\chi_\alpha^2}{p(n-1)}.$$

При обробці результатів ранжування може виникнути необхідність визначення залежності між двома групами об'єктів або двома ранжуваннями однієї групи за двома критеріями або між відповідними ранжуваннями двох експертів. У таких випадках можна застосувати коефіцієнти парної рангової кореляції Спірмена і Кендалла.

Коефіцієнт кореляції Спірмена визначається за формулою:

$$\rho = 1 - \frac{6}{n^3 - n} \sum_{i=1}^{n} \left(a_{1i} - a_{2i}\right)^2 \qquad (1.2)$$

де n – кількість об'єктів, що ранжуються;

a_{1i} та a_{2i} – ранги у першому та другому ранжуванні відповідно.

Коефіцієнт парної рангової кореляції Спірмена приймає значення від -1 до +1. При ρ = 1 відповідає повної узгодженості

думок експертів; при ρ = -1 вказує на факт протилежності думок експертів.

При зв'язних рангах коефіцієнт Спірмена набуває вигляду:

$$\widetilde{\rho} = \frac{\rho + T_1 + T_2}{\sqrt{(1 - T_1)(1 - T_2)}},$$

де ρ – коефіцієнт рангової кореляції Спірмена за формулою (1.2), а величини Т1 та Т2 обчислюють відповідно як

$$T_1 = \frac{3}{n^3 - n} \sum_{k_1} k_1 (k_1 - 1) \quad T_2 = \frac{3}{n^3 - n} \sum_{k_2} k_2 (k_2 - 1);$$

де k1 та k2 – кількість різних зв'язних рангів у першому та другому ранжуванні відповідно.

Коефіцієнт рангової кореляції Кендалла визначається за формулою:

$$\tau = \frac{4k}{n(n-1)} - 1$$

де k – кількість пар рангів з прямим порядком;

n – кількість об'єктів, для яких визначається рангова кореляція.

Коефіцієнт τ приймає значення від -1 до +1. Рівність τ = 1 вказує на точну лінійну кореляцію.

Таким чином, оцінка думок експертів за допомогою коефіцієнтів конкордації і кореляції при колективному прийнятті рішень дозволяє виявити і об'єднати різні підходи експертів до оцінки різних явищ (ознак, критеріїв), що дозволяє провести поглиблений аналіз ситуації і прийняти більш зважене обґрунтоване рішення.

Ухвалення рішення завжди здійснюється з метою отримання певного результату. Якість цього результату залежить від повноти обліку всіх факторів, які є істотними для наслідків від прийнятих рішень. При цьому також необхідно враховувати, що більшість реальних завдань доводиться вирішувати в умовах невизначеності – неповноти, недостовірності і несвоєчасності надходження інформації [26].

1.3.4 Класифікація контрзаходів

Спираючись на роботу [27] можна виділити наступні типи контрзаходів:

За часом застосування контрзаходи можуть поділятися:

– на попереджувальні, які здатні повністю попередити наступ можливого ризику, при цьому їх застосування здійснюється до настання ризикової події;

– контрзаходи реакції на ризик, що призначені для часткової компенсації ризику і застосовуються після настання небажаних подій.

При цьому витрати на реалізацію попереджувальних контрзаходів стягуватимуться незалежно від того настало чи ні подія неуспіху на відміну від контрзаходів реакції на ризик, витрати на реалізацію яких потрібні тільки в разі настання ризику. З іншого боку застосування останніх не завжди повністю може компенсувати ризик, що, в кінцевому рахунку, може призвести до збільшення залишкового ризику і для медичних виробів самостійно застосовуються вкрай рідко.

За об'єктом впливу контрзаходи можуть поділятися на такі типи:

– скорочення або виключення первинних загроз, дефектів і помилок в системі дослідження, обумовлених недоліками їх проєктування, розробки, модифікації або планування, що

відбиваються на ризиках функціональної придатності або характеристиках системи;

– скорочення уразливості компонентів системи при впливі на них загроз, дефектів і помилок, шляхом введення засобів захисту для блокування їх можливого негативного впливу на ризики функціонування;

– безпосереднє зміна та скорочення наслідків проявів ризиків шляхом їх оперативного виявлення і ліквідації збитку.

Необхідно відзначити, що виключення первинних загроз і скорочення вразливостей призводить в кінцевому підсумку до поліпшення самої системи, тоді як скорочення наслідків ризиків відбувається зазвичай при збереженні первинних джерел і причин, що їх викликають.

Для прикладу в таблиці 1.6 наведено співвідношення контрзаходів різних типів, де в комірках зазначені ранги критичних ризиків, що були визначені в таблиці 1.5, для яких застосування контрзаходів даного типу доцільніше.

Таблиця 1.6 – Співвідношення контрзаходів різних типів

За часом застосування	*За об'єктом впливу*		
	Виключення первинних загроз	Скорочення уразливості	Скорочення наслідків
Попереджувальні контрзаходи	I	I	–
Контрзаходи реакції на ризик	–	II	II

1.4 Модель верифікації та перевірки для біомедичних застосувань

Медичні експертні системи корисні та необхідні, оскільки вони допомагають лікарям у прийнятті рішень щодо плану лікування. Таку систему можна розробити, застосувавши ефективні методики проєктування з метою підвищення продуктивності. Оскільки медичні експертні системи залежать від домену, аналіз вимог повинен бути проведений до розробки програмного забезпечення. У об'єктно-орієнтованому дизайні основну роль відіграють схеми використання інциденту. Аналіз схем випадків використання можна проводити шляхом більш ефективного аналізу ієрархії класів. Аналіз випадків за допомогою методу нарізки ідентифікує деталі одного рівня ієрархії класів.

Медичні експертні системи залежать від домену і мають два основних компоненти, а саме базу знань і механізм логічного виведення. Медичні додатки можуть бути розроблені за допомогою рішень, прийнятих експертними системами. Для розробки таких додатків проєктування та впровадження програмного забезпечення є важливими завданнями після аналізу вимог. Тому дизайн програмного забезпечення є важливою діяльністю в життєвому циклі розробки програмного забезпечення. У минулому було запропоновано багато дизайнерських показників, які включали високу згуртованість і низьке зчеплення. Однак автоматизований дизайн зі схем варіантів використання можуть забезпечити швидкі методи проєктування за допомогою структурованих схем розробки програмного забезпечення, таких як діаграми класів, схеми активності, схеми послідовностей та схеми розгортання. Наприклад, діаграми класів можна намалювати, знаючи ієрархію успадкування, яку можна отримати з специфікації вимог.

З іншого боку, послідовні діаграми допомагають відобразити послідовність завдань, що допоможе керівникам проєктів програмного забезпечення оптимізувати час і ресурси. Тому

необхідно мати автоматичний програмний метод, який може забезпечити дизайн схем і дизайн документів з необхідною якістю. Останнім часом схеми потоку даних замінюються методами уніфікованої мови моделювання (UML) для виконання об'єктно-орієнтованого аналізу та дизайну. Така конструкція обмежена багатьма об'єктно-орієнтованими метриками, включаючи абстракцію, модульність, глибину успадкування, кількість нащадків на клас, зважені методи для класу, рівень згуртованості і зчеплення. Крім того, живописна репрезентативна кількість всіх символів, діаграм з текстом в UML дозволяє краще зрозуміти діяльність, процес проєкта, що знаходиться в розробці [28].

У [29] пропонується автоматизована модель нарізки для об'єктно-орієнтованої конструкції, яка корисна для проєктування медичних експертних систем. Ця запропонована модель використовує методи генерації графіків і нарізки, запропоновані Jaiprakash та іншими [29] для ефективної нарізки варіантів використання. Крім того, ця модель допомагає лікарям для надання плану лікування хворим на астму. Таким чином, він посилює схеми варіантів використання за допомогою штучного інтелекту, застосовуючи правила для ефективного прийняття рішень.

Модуль генерації та нарізки графіка складається з двох етапів, а саме фази ідентифікації та фази нарізки. Етап ідентифікації виявляє схеми випадків використання, що відносяться до програми, і допомагає вивести їх з сховища випадків використання. Фаза нарізки використовує існуючий підхід генерації графіків, запропонований Jaiprakash та іншими [29], щоб нарізати схему використання регістру на акторів і варіанти використання, а варіанти використання застосовуються для генерації графіка. Для демонстрації запропонованої роботи розроблено та впроваджено медичну реєстраційну заявку з використанням мови моделювання UML та запропонованої моделі. У модулі медичної реєстрації зберігаються реквізити

пацієнта, дані лікаря та реквізити ліків та лікування. Зберігається база знань, що складається із загальних правил вирішення проблем та правил домену для пошуку та лікування астми. Цю базу знань можна оновити, додавши більше правил за допомогою класифікатора через навчання. У [30] для аналізу розглядається схема варіантів використання, присутня в репозиторії варіантів використання.

З іншого боку, надійність біомедичних застосувань є одним з найбільш критичних параметрів. Акцент робиться на аналізі даних про відмову. Більшість моделей надійності програмного забезпечення є аналітичними моделями, отриманими з припущень про те, як відбуваються збої. Акцент робиться на припущеннях моделі та інтерпретації параметрів.

Для того, щоб розробити корисну модель надійності програмного забезпечення та судити при використанні моделей, потрібне поглиблене розуміння того, як виробляється програмне забезпечення; як помилкові дані вводяться, як програмне забезпечення тестують, як виникають помилки, типи помилок, фактори навколишнього середовища можуть допомогти розробникам у виправданні розумності припущень, оцінити корисність моделі [31].

Для використання даних, отриманих під час компіляції варіанту використання, для підвищення надійності біомедичних застосувань можна використовувати V-модель. V-модель – це модель життєвого циклу розробки програмного забезпечення, де виконання процесів відбувається послідовно у V-формі [32]. Вона також відома як модель верифікації та перевірки.

V-модель є розширенням моделі водоспаду і базується на об'єднанні етапу тестування для кожного відповідного етапу розробки. Це означає, що для кожного етапу циклу розробки існує безпосередньо пов'язаний етап тестування. Це високодисциплінована модель і наступний етап починається тільки після завершення попереднього етапу.

За V-моделлю паралельно планується відповідний етап тестування етапу розробки. Таким чином, є етапи верифікації на одній стороні етапів "V" та перевірки з іншого боку. Етап кодування приєднується до двох сторін V-моделі.

На наступній ілюстрації зображені різні фази у V-моделі [32].

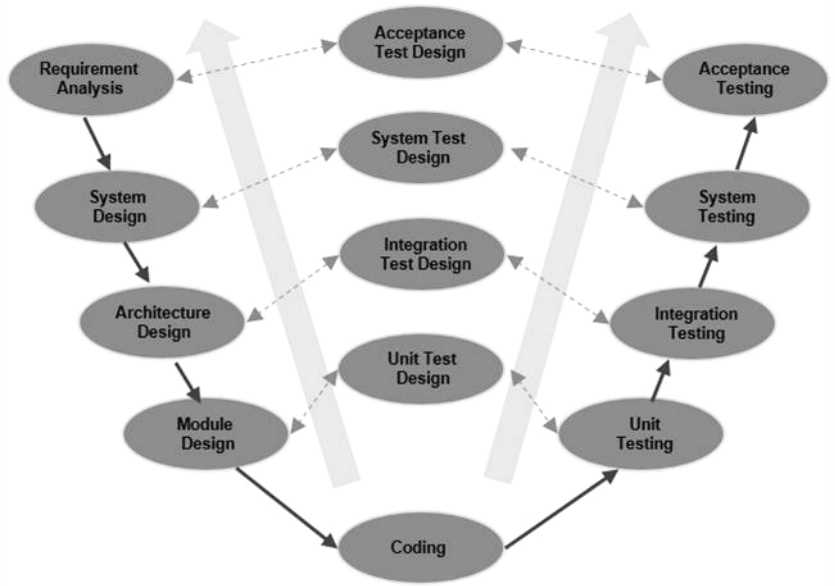


Рисунок 1.11 – Етапи життєвого циклу за V-моделлю [32]

Застосування V- моделі майже таке ж, як і моделі водоспаду, так як обидві моделі мають послідовний тип. Вимоги повинні бути дуже чіткими до початку проєкту, тому що зазвичай дорого повернутися назад і внести зміни. Дана модель використовується в галузі медичного розвитку, так як це строго дисциплінований домен.

Перевага методу V-моделі полягає в тому, що його дуже легко зрозуміти і застосувати. Простота цієї моделі також полегшує управління. Недоліком є те, що модель не гнучка до змін, тому, якщо є вирогідність зміни вимог, що є дуже поширеною

практикою в сучасному динамічному світі, зробити зміни стає дуже дорого.

Перевагами методу V-моделі є:

1. Це високодисциплінована модель, в якій фази завершуються по одній.
2. Добре працює для невеликих проєктів, де вимоги дуже добре розуміються.
3. Проста у розумінні та використанні.
4. Легкість керування завдяки жорсткості моделі. Кожен етап має конкретні кінцеві результати і процес розгляду.

Недоліки методу V-моделі наступні:

1. Високий ризик і невизначеність.
2. Не дуже хороша модель для складних і об'єктно-орієнтованих проєктів.
3. Погана модель для довгих і поточних проєктів.
4. Не підходить для проєктів, де вимоги мають помірний або високий ризик зміни.
5. Після того, як додаток надходить на стадію тестування, важко повернутися і змінити функціональність.
6. Протягом життєвого циклу не виробляється жодне робоче програмне забезпечення.

1.5 Контрольні запитання

1. 1.Що таке медичні інформаційні системи?
2. Як класифікцють МІС?
3. Яке призначення медичних консультативно-діагностичних систем?
4. Яке призначення медичних консультативно-діагностичних систем?
5. Яке призначення медичних приладо-комп'ютерних систем?
6. Яке призначення медичних інформаційних систем рівня лікувально-профілактичних установ?

7. Які основні рівні автоматизації медичних інформаційних систем?

8. Наведіть визначення ризику.

9. Які методи експертного оцінювання використовуються?

10. Як узгоджуються рішення експертів?

11. Яка модель використовується для керування якістю медичних та телемедичних систем?

1.6 Література до розділу

1. Україна 2030е — країна з розвинутою цифровою економікою. Електроний ресурс. Онлайн:https://strategy.uifuture.org/kraina-z-rozvinutoyu-cifrovoyu-ekonomikoyu.html

2. Грекул В.И. Проектирование информационных систем [Електронний ресурс] / В.И. Грекул - Режим доступа: http://www.intuit.ru/department/se/devis/

3. Избачков Ю.С. Информационные системы: учебник для ВУЗов./ Ю.С. Избачков, В.Н. Петров. – СПб.:Питер, 2006. – 656с.

4. Назаренко Г.И. Медицинские информационные системы: теория и практика / Г.И. Назаренко, Я.И. Гулиев, Д.Е. Ермаков. – М.: Физматлит, 2015. – 320 с.

5. Деньга, А. В. Медицинская информационная система для автоматизации рабочего места врача / А. В. Деньга, В. В. Шлыков // Молодой ученый. — 2016. — № 13 (117). — С. 108-110.

6. Гогина, О. А. Основные стандарты и модели интеграции медицинских информационных систем / О. А. Гогина. // Молодой ученый. — 2017. — № 18 (152). — С. 8-11.

7. Каюмова, Д. Д. Развитие информационной системы здравоохранения / Д. Д. Каюмова. // Молодой ученый. — 2019. — № 3 (241). — С. 119-121.

8. Болгов М.Ю. Автоматизация медицинских учреждений: Руководство пользователя TherDep5 / М.Ю. Болгов. – К.: Куприянова, 2016. – 464 с.

9. Титаренко Г. А. Информационные технологии управления / Г. А. Титаренко. – М.: ЮНИТИ-ДАНА, 2008.

10. ДСТУ ISO 14971:2009 Вироби медичні. Настанови щодо управління ризиком (ISO 14971:2007, IDT)

11. Технічний регламент щодо медичних виробів. Затверджено постановою Кабінету Міністрів України від 2 жовтня 2013р. №753.

12. ISO/IEC Guide 51:1999 Safety aspects – Guidelines for the inclusions in standards (Аспекти безпеки. Настанови щодо включення в стандарти).

13. IEC/TR 60513:2.0 Fundamental Aspects of Safety Standards for Medical Electrical Equipment (Фундаментальні аспекти стандартів з безпеки для електричного медичного обладнання).

14. ДСТУ 4659-1:2006 Клінічні дослідження медичних виробів для людей. Частина 1: Загальні вимоги (ISO 14155-1:2003, MOD).

15. ДСТУ 4659-2:2006 Клінічні дослідження медичних виробів для людей. Частина 2. Плани клінічного дослідження (ISO 14155-2, MOD).

16. IEC 60300-3-9:1995 Dependability management – Part 3. Application guide – Section 9. Risk analysis of technological systems (Управління надійністю. Частина 3: настанови щодо застосування. Секція 9: Аналіз ризиків технологічних систем).

17. IEC 61882 Hazard and Operability Studies (HAZOP studies) – Application guide (Дослідження небезпеки та працездатності (HAZOP). Настанови із застосування).

18. Ветошкин А.Г. Надежность и безопасность технических систем / А.Г. Ветошкин, В.И. Марунин, под.ред. А.Г. Ветошкина. – Пенза: Изд-во Пенз. гос. ун-та. – 2002. – 129 с.

19. IEC 60812 Analysis techniques for system reliability – Procedures for failure mode and effects analysis (FMEA) (Методи аналізування надійності системи. Процедури для аналізування характеру та наслідків відмов (FMEA).

20. IEC 61025 Fault tree analysis (FTA) (Аналіз дерева помилок (FTA).

21. Hazard Analysis and Critical Control Points and Application Guidelines, Adopted, August 14, 1997? National Advisory Committee on Microbiological Criteria for Foods: http://vm.cfsan.fda.gov/~comm/nacmcfp.html (Принципи та керівні положення щодо застосування. Система аналізування ризиків та критичних контрольних точок (HACCP), ухвалено 14 серпня 1997 р. Національний консультативний комітет щодо мікробіологічних критеріїв стосовно харчових продуктів.

22. Рябинин И.А. Надежность и безопасность структурно-сложных систем / И.А. Рябинин. – СПб.: Политехника. – 2000. – С.11–12.

23. Крючковский В.В. Интроспективный анализ. Методы и средства экспертного оценивания: Монография / В.В. Крючковский, Э.Г. Петров, Н.А. Соколова, В.Е. Ходаков ; под ред. Э. Г. Петрова. – Херсон : Гринь Д.С. – 2011. – 168 с.

24. Миронова Наталя Олексіївна. Методи та Інформаційна технологія багатокритеріального колективного експертного оцінювання з використанням нечітких оцінок: Дис. канд. техн. наук: 05.13.06 / Запорізький національний технічний ун-т. – Запоріжжя. – 2016. – 159 с.

25. Крючковский В.В. Оценка степени согласованности мнений экспертов при коллективном принятии решений / В.В.Крючковский // Системні технології. Регіональний міжвузівський збірник наукових праць. – 2009. – Вип. 6(65). Дніпропетровськ : Національна металургійна академія, 2009. – С. 65-74.

26. Авраменко В.П. Принятие управленческих решений в условиях неопределенности и нечеткости исходной информации / В.П. Авраменко, В.Ф. Ткаченко, Л.Б. Середа // Радіоелектроніка, інформатика, управління. –2010. – № 2. – С. 101-106.

27. Модельно-орієнтоване проектування випробувань складних технічних обєктів / О.В. Шитіова, С.Б. Бєліков, Г.В. Табунщик. – Запоріжжя:Дике Поле. 2017. – 170с.

28. Проектування та моделювання програмного забезпечення сучасних інформаційних систем: навчальний

посібник / Г. В. Табунщик, Т.І. Каплієнко, О.А Петрова − Запоріжжя: Дике поле, 2016. – 270 с.

29. Jaiprakash TL, Mall R. A dynamic slicing technique for UML architectural mod-els. IEEE Transact Software Eng 2011; 37: 735- 771.

30. Lalitha R. A software design technique for developing medical expert systems through use case analysis [Електронний ресурс] / R. Lalitha, B. Latha, G. Sumathi – Режим доступу до ресурсу: https://www.biomedres.info/biomedical-research/a-software-design-technique-for-developing-medical-expert-systems-through-use-case-analysis.html.

31. Pham H. Software reliability models for critical applications [Електронний ресурс] / H. Pham, M. Pham – Режим доступу до ресурсу: https://www.osti.gov/biblio/10105800.

32. SDLC - V-Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm.

33. Шитикова Е. В. Модельно-ориентированные методы автоматизации процесса испытаний сложных технических систем / Шитикова Е. В. Табунщик Г. В. // Электротехнические и компьютерные системы № 22 (98), 2016, -С. 338 – 346 http://dx.doi.org/10.15276/eltecs.22.98.2016.61

34. Tabunshchyk G. Flexible Technologies for Smart Campus/ D. Van Merode, G. Tabunshchyk, K. Patrakhalko, Y. Goncharov // Proceedings of XIII International Conference on Remote Engineering and Virtual Instrumentation (REV2016) (24-26 February, 2016, Madrid, Spain) UNED: pp. 58-62.

35. А.С. № 66615 Система керування контентом для віддалених експериментів з дослідження надійності вбудованих систем /Табунщик Г.В., Охмак В.О., опубл. 13.07.2016

36. Tabunshchyk G. Multipurpose Educational System based on Raspberry Pi/ G. Tabunshchyk, D. Van Merode , O.Petrova, V. Okhmak // Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering, Nitra, Slovakia, 12-15 September, 2016 – pp. 202-206

37. Tabunshchyk G. Smart-campus infrastructure development based on BLE 4.0/ G. Tabunshchyk, PhD., D. Van

Merode, Y. Goncharov, K. Patrakhalko // Journal Electrotechnic and Computer Systems No. 18 (94), 2015 17 – 20

38. Arras, P. Iterative Pattern for the Embedding of Remote Laboratories in the Educational Process / P. Arras, K. Henke, G.Tabunshchyk, D. V.Merode // 12th International Conference on Remote Engineering and Virtual Instrumentation (REV 2015) 25-28 February 2015, Bangkok, Thailand, PP. 52-55 (10.1109/REV.2015.7087262)

ЧАСТИНА 2. ОСНОВНІ ПОНЯТТЯ ПРОЄКТУВАННЯ АРХІТЕКТУРИ МЕДИЧНИХ ТА ТЕЛЕМЕДИЧНИХ СИСТЕМ

Ефективне впровадження новітніх результатів розвитку науки і техніки в практику охорони здоров'я можливе за умови створення системи та інфраструктури трансляційної медицини, що дозволить об'єднати в єдине ціле всі життєві етапи медичної технології від її розробки до використання в клінічній практиці. Істотним стримуючим фактором є недостатнє застосування сучасних інформаційних та комунікаційних технологій і впровадження методів систематизації медичної інформації. Розвиток телемедицини стримують також і проблеми в галузі інформаційної безпеки, які пов'язані з дотриманням вимог захисту особистих даних пацієнтів і персональних даних про стан їх здоров'я.

Спеціалізована медична апаратура, а також цифрові зображення, створені за її допомогою, стали невід'ємною частиною діагностичного та лікувального процесів. Так само активно використовуються цифрові можливості для передачі і обміну медичними зображеннями у комп'ютерних мережах. Електронний обмін діагностичними зображеннями забезпечує розвиток і поширення телемедицини з використанням економічно виправданих, технічно і організаційно оптимальних сеансів віддаленого консультування, що вимагає швидкої і надійної системи передачі даних.

Розроблення програмного забезпечення підпорядковане певному життєвому циклу. ***Життєвий цикл*** – це впорядкований набір видів діяльності, здійснюваний та керований у рамках кожного проєкту з розробки програмного забезпечення (ПЗ).

На узагальненому рівні життєвий цикл (ЖЦ) може містити лише три етапи:

– аналіз;
– проєктування;
– реалізація.

Етап аналізу концентрується на системних вимогах. Вимоги визначаються та специфікуються. Здійснюється розроблення та інтеграція функціональних моделей та моделей даних для системи. Крім того, фіксуються нефункціональні вимоги й інші системні обмеження.

Етап проєктування поділяється на два основні підетапи: архітектурне та деталізоване проєктування. Зокрема, проводиться уточнення конструкції програми для архітектури клієнт/сервер, що інтегрує об'єкти інтерфейсу користувача та бази даних. Ставляться та фіксуються питання проєктування, що впливають на зрозумілість, пристосованість до супроводу та масштабованість системи.

Етап реалізації містить створення коду клієнтських програмних додатків і серверів баз даних.

Коротко кажучи, аналіз вказує на те, що робити; проєктування – на те, як за допомогою наявної технології зробити це, ареалізація втілює задумане на попередніх етапах.

Спираючись на вищесказане, на деталізованому рівні ЖЦ можна розділити на наступні етапи:

– концептуалізація системи;
– специфікація вимог;
– проєктування архітектури системи;
– деталізоване проєктування;
– реалізація;
– інтеграція;
– супровід.

Концептуалізація системи. Розроблення програмного забезпечення починається з бізнес-аналітиків або користувачів, які придумують додаток і формулюють первинні вимоги до нього. Результатом цього етапу є документ, що містить викладення вимог. Це більшою мірою текстовий документ з деякими неформальними діаграмами та таблицями. Цей документ, як правило, не має формальних моделей, винятком можуть бути певні прості та широко відомі нотації, що можуть бути легко сприйняті замовниками.

Специфікація вимог. На даному етапі *аналітик* ретельно досліджує та безжалісно переформульовує вимоги, конструюючи моделі, виходячи з концепцій системи. Він повинен тісно працювати із замовником, щоб досягти розуміння завдання, тому що визначення, отримані на попередньому етапі, рідко виявляються повними або коректними. ***Аналітична модель*** – це стисла й точна абстракція того, що саме повинна робити система. Аналітична модель не повинна містити ніяких рішень щодо реалізації. Наприклад, клас *Window* в аналітичній моделі системи керування вікнами для робочої станції повинен бути описаний у термінах видимих атрибутів і операцій.

Аналітична модель складається з двох частин: моделі предметної області (*domain model*) – опису об'єктів реального світу, що відображає система, та моделі програмного додатка (*application model*)– опису видимих користувачеві частин самого додатка. Наприклад, для додатка біржового маклера об'єктами предметної області можуть бути акції, облігації, торги й комісія. Модель предметної області, у свою чергу, складається з моделі класів та моделі взаємодії. Об'єкти моделі додатка можуть керувати здійсненням торгів і відображати результати. Гарна модель повинна бути доступною для розуміння та критики з боку експертів, які не є програмістами.

Проєктування архітектури системи. Команда розробників продумує стратегію вирішення завдання на вищому рівні,

визначаючи архітектуру системи. На цьому етапі визначаються концепції, що послужать основою для прийняття рішень на наступних етапах проєктування. Проєктувальник системи повинен вибрати параметри системи, відповідно до яких буде проводитися оптимізація, запропонувати стратегічний підхід до завдання, провести попередній розподіл ресурсів. Наприклад, проєктувальник може вирішити, що будь-які зміни зображення на екрані робочої станції повинні бути швидкими та плавними, навіть при переміщенні та закритті вікон. На підставі цього рішення він може вибрати відповідний протокол обміну та стратегію буферизації пам'яті.

Проєктування класів. Проєктувальник класів уточнює аналітичну модель відповідно до стратегії проєктування системи. Він проробляє об'єкти предметної області й об'єкти моделі додатка, використовуючи однакові об'єктно-орієнтовані концепції та позначення, незважаючи на те, що ці об'єкти лежать у різних концептуальних площинах. Мета проєктування класів полягає в тому, щоб визначити, які структури даних й алгоритми потрібні для реалізації кожного класу. Наприклад, проєктувальник класів повинен вибрати структури даних і алгоритми для всіх операцій класу Window.

Реалізація. Відповідальні за реалізацію займаються перекладом класів і відносин, що утворилися на попередньому етапі, на конкретну мову програмування, втіленням їх у базі даних або в апаратному забезпеченні. Ніяких ускладнень на цьому етапі бути не повинно, тому що всі відповідальні рішення вже були прийняті на попередніх етапах. У процесі реалізації необхідно використовувати технології розроблення програмного забезпечення, щоб відповідність коду проєкту була очевидною, а система залишалася гнучкою та розширюваною. У нашім прикладі група реалізації повинна написати код класу Window будь-якою мовою програмування, використовуючи виклики функцій або методи графічної підсистеми робочої станції.

Об'єктно-орієнтовані концепції діють протягом усього життєвого циклу програмного забезпечення, на етапах аналізу, проєктування й реалізації. Ті самі класи будуть переходити від одного етапу до іншого без будь-яких змін у нотації, хоча на останніх етапах вони істотно обростуть деталями.

Етап інтеграції. Інтеграція модулів ретельно планується спочатку ЖЦ ПЗ. Програмні компоненти для окремої реалізації повинні бути ідентифіковані на ранніх стадіях аналізу системи. Порядок реалізації повинен дозволяти якомога плавнішу збільшувану інтеграцію.

Головна трудність полягає в існуванні взаємних зворотних залежностях між модулями. Хороший проєкт системи відрізняється мінімальною зв'язаністю (*low coupling*) модулів. Все одно, час від часу виникають взаємозалежні модулі, що не можуть функціонувати ізольовано.

Об'єктно-орієнтовані системи повинні бути спроєктовані під інтеграцію. Кожний модуль повинен бути як найбільш незалежним. Залежності між модулями необхідно ідентифікувати та мінімізувати на етапах аналізу та проєктування. Якщо система спроєктована неякісно, етап інтеграції призведе до хаосу та поставить під загрозу весь проєкт.

Етап супроводження. Цей етап наступає після успішного постачання замовникові готової програмної системи.

Супроводження складається з трьох стадій:

– підтримання експлуатації;
– адаптивне супроводження;
– супроводження для поліпшення.

Підтримання експлуатації зв'язана з рутинними завданнями супроводу, що необхідні для підтримки системи в стані готовості до використання користувачами та експлуатаційним персоналом. *Адаптивний супровід* зв'язаний з відстеженням та аналізом роботи системи, налагодженням її функціональних можливостей відносно змін зовнішнього середовища та адаптацією системи для

досягнення заданої продуктивності та пропускної спроможності. Під *супроводом для поліпшення* розуміють перепроєктування та модифікацію системи для задоволення нових або суттєво змінених вимог.

Коли супровід проєкту стає недоцільним, його слід згорнути.

Концепції індивідуальності, класифікації, поліморфізму та спадкування діють протягом усього процесу розробки.

Усі ці етапи можна використати як для водоспадного процесу, де ці етапи повинні виконуватися в зазначеному вище порядку, так і для ітераційного підходу, в якому кожна складова частина розробляється вдекілька етапів.

Деякі класи не входять до аналітичної моделі. Вони з'являються пізніше, на етапах проєктування або реалізації. Наприклад, структури даних, подібні до дерев, хеш-таблиць і зв'язних списків, рідко з'являються в реальному світі та зазвичай невидимі для користувачів. Проєктувальники додають їх у систему для того, щоб забезпечити підтримку обраних алгоритмів. Об'єкти структур даних існують усередині комп'ютера та не є безпосередньо спостережуваними.

Тестування як окремий етап ЖЦ не розглядається, тому що воно дуже важливе, але повинне бути частиною системи контролю якості, що застосовується протягом усього ЖЦ. Розробники повинні порівнювати аналітичні моделі з реальністю, перевіряти проєктувальні моделі на наявність помилок різних видів, а не тільки тестувати коректність реалізації. Виділення всіх операцій з контролю якості в окремий етап коштує дорожче та є менш ефективним.

2 КОНЦЕПТУАЛІЗАЦІЯ СИСТЕМИ

2.1 Розроблення концепції системи

Концептуалізація системи – це зародження додатка.

У більшості випадків ідеї, на яких ґрунтуються нові системи, є продовженням уже існуючих ідей.

Існує кілька способів пошуку концепцій нових систем:

– нова функціональність – можна додати функціональність в існуючу систему;

– модернізація – зняття обмежень або універсалізація роботи системи;

– спрощення – надання звичайним людям можливості займатися тим, чим раніше займалися тільки фахівці;

– автоматизація ручних процесів;

– інтеграція – об'єднання функціональності різних систем;

– аналогії – пошук аналогій в інших предметних областях і дослідження їх на наявність корисних ідей;

– глобалізація –вивчення культури й ділової практики інших країн та впровадження їх досвіду.

Перед тим як вкладати кошти та час у розробку, потрібно оцінити можливість створення системи, витрати й ризики, пов'язані з її розробленням, потребу в системі та відношення виграшу до витрат.

Робота над системою починається з розроблення концепції. Гарна концепція повинна містити наступну інформацію:

– для кого призначений додаток – визначаються зацікавлені особи, які є 2-х типів: спонсори та користувачі.

– яке завдання буде вирішувати додаток – визначаються функції, які буде мати система;

– де буде використовуватися система – визначається де буде використовуватися система, чи буде вона незалежною, чи буде доповнювати вже існуючі системи;

– коли буде потрібна система. Для нового додатка важливо два аспекти, зв'язаних з часом. По-перше, це час за який система може бути розроблена з урахуванням обмежень щодо вартості та ресурсів. По-друге, це час, за який система може бути розроблена, щоб задовольнити вимоги бізнесу. Треба переконатися, що оцінка часу, отримана з урахуванням технологічних можливостей, відповідає потребі бізнесу;

– чому потрібна система – підготовлюють економічне обґрунтування системи;

– як буде працювати система – розглядаються можливості використання різних архітектур.

Наприклад, потрібно розробити систему «Банкомат».

Концепція: необхідно розробити програмне забезпечення, що дозволить клієнтам одержувати доступ до банківської комп'ютерної системи та виконувати операції без участі банківських службовців.

1. Для кого призначений додаток?

Банкомати виробляються кількома компаніями. Тому тільки виробник автомата може компенсувати витрати на створення програмного забезпечення банкомата.

2. Яке завдання буде вирішувати додаток?

ПЗ банкомата повинне працювати як на банк, так і на клієнта. З погляду банку, програмне забезпечення підвищує рівень автоматизації та скорочує обсяг ручної роботи.

З погляду клієнта, банкомати повинні бути поширені повсюдно та цілодобово доступні.

ПЗ повинне бути простим у використанні та зручним. Система повинна бути надійною та захищеною, оскільки вона працює з грошима.

3. Де буде використовуватися система?

Актуальний засіб для всіх фінансових установ

4. Чому потрібна система?

Новий продукт дозволить виробнику бути більш конкурентоспроможним.

5. Як буде працювати система?

Планується реалізувати трирівневу архітектуру, відокремивши інтерфейс користувача від програмної логіки, а логіку від бази даних.

Після того як ідея кристалізується, формулюються вимоги до системи, що будуть описувати мету та загальний підхід до створення системи.

2.2 Встановлення вимог

Мета встановлення вимог полягає в тому, щоб дати розгорнуте визначення функціональних, а також нефункціональних вимог, які учасники проєкту очікують затвердити в реалізованій і розгортуваній системі.

Вимоги визначають послуги, очікувані від системи (формулювання сервісів) та обмеження, яким система повинна підлягати (формулювання обмежень). Ці формулювання сервісів можна об'єднати в кілька груп: одна із груп описує межі системи, інша – необхідні бізнес-функції (функціональні вимоги), а третя – необхідні структури даних (вимоги до даних).

Вимоги необхідно одержати від зацікавлених осіб. Цей вид діяльності називається *виявленням вимог* і здійснюється аналітиком бізнес-процесів (або системним аналітиком).

До зацікавлених осіб в проєкті відносяться:

- Замовники
- Користувачі
- Аналітики вимог
- Розробники
- Тестери
- Технічні письменники
- Менеджер проєкту
- Співробітники юридичного відділу
- Представники промислових організацій
- Співробітники відділу продажів

Потрібно виконати ретельний аналіз зібраних вимог для виявлення в них повторів і протиріч. Це незмінно приводить до перегляду вимог та повторного їх узгодження з зацікавленими особами.

Вимоги, задовільні з погляду зацікавлених осіб, документуються. При цьому вимогам дають визначення, класифікують, нумерують і привласнюють їм пріоритети. Структура документа, що описує вимоги, відповідає шаблону, обраному в організації для цієї мети.

Хоча документ, що фіксує вимоги, носить значною мірою описовий характер, цілком можливо включити до нього високорівневу схематичну бізнес-модель. Як правило, *бізнес-модель* складається з моделі меж системи, моделі бізнес-прецедентів і моделі бізнес-класів.

2.2.1 Виявлення вимог

Бізнес-аналітик виявляє вимоги до системи за допомогою консультацій, до участі в яких залучаються замовники, користувачі й експерти в проблемній області. У деяких випадках бізнес-аналітик має достатній досвід у проблемній області, і допомога експерта може не знадобитися. У цьому випадку бізнес-аналітик являє собою різновид експерта проблемної області.

Завдання бізнес-аналітика полягає в тому, щоб об’єднати два набори вимог у *бізнес-моделі*. Бізнес-модель містить *модель бізнес-класів* і *модель бізнес-прецедентів*. *Модель бізнес-класів* – це діаграма класів верхнього рівня, що ідентифікує й зв’язує між собою бізнес-об’єкти. *Модель бізнес-прецедентів* – це діаграма прецедентів верхнього рівня, що ідентифікує основні функціональні будівельні блоки системи.

Методи виявлення вимог розподіляються на традиційні та сучасні.

До традиційних належать:

– інтерв’ювання;

– анкетування;

– спостереження;

– вивчення документів програмних систем.

До сучасних:

– прототипування;

– спільне розроблення програмних додатків (JAD-метод);

– швидке розроблення програмних додатків (RAD-метод).

Використання ***інтерв’ю*** являє собою основний *метод для* збору інформації. Більшість інтерв’ю проводяться із зацікавленими особами, інтерв’ю з якими дозволяють виявити більшою мірою вимоги, що випливають із прецедентів. Якщо бізнес-аналітик не має достатнього досвіду в проблемній області, можна також про інтерв’ювати відповідних експертів.

Проблеми інтерв'ювання:

- замовник може не знати, чого хоче;
- не хоче співпрацювати;
- не виражає вимог.

Існує 2 основних види інтерв'ю:

1. Структуроване (формальне) – слід заздалегідь підготувати питання, 2 категорій: питання з відкритим безліччю відповідей і питання із замкнутим безліччю відповідей;
2. Неструктурована (не формальне).

Використання анкет або ***анкетування*** (*questionnaires*) – ефективний спосіб збору інформації від багатьох замовників. Звичайно анкети використовуються додатково до інтерв’ю, а не замість них. Виняток можуть становити проєкти з низьким ризиком, цілі яких ясно окреслені. Для таких проєктів звичайно досить використати анкети із запитаннями, що носять пасивний характер і не відрізняються великою глибиною. . В анкетуванні найчастіше використовуються питання з замкнутим списком відповідей.

У загальному випадку, анкетування менш продуктивне, ніж використання інтерв'ю, оскільки до питань або можливих відповідей не можна внести додаткову ясність.

Питання можуть приймати форму:

– багатоальтернативні питання – при відповіді на ці питання респондент повинен вказати один або більше відповідей, вибравши їх із прикладеного списку. Крім того, іноді допускаються додаткові коментарі до питань з боку респондента.

– рейтингові питання – при відповіді на цей тип питань респондент повинен висловити свою думку щодо висловленого твердження. Для цього можуть бути використані такі рейтингові значення як: абсолютно згоден, згоден, ставлюся нейтрально, не згоден, абсолютно не згоден, не знаю.

– питання з ранжируванням – цей тип питань передбачає ранжування відповідей за допомогою привласнення ним послідовних номерів – процентних значень і використання інших засобів упорядкування.

Розрізняють 2 види ***спостереження***: активне, яке полягає у безпосередньому зануренні в середу (аналітик працює, як учасник команди, що дозволяє поліпшити розуміння процесів) та пасивне, яке полягає у вивченні документів або у розмовах з експертами.

Вивчення документів і програмних систем є неоціненним методом виявлення як вимог типу прецедентів, так і вимог, зв'язаних зі знанням проблемної області. Цей метод використовується завжди, хоча він може стосуватися тільки окремих сторін системи.

Таблиця 2.1 – Вивчення документів і програмних систем

Організаційні документи	**Системні форми і звіти**
1. Форми ділових документів (по можливості – заповнені)	1. Системні моделі аналізу і проєктування
2. Опис робочих процедур	2. Звіти разом з документацією
3. Посадові обов'язки	3. Системні керівництва по експлуатації
4. Методичні посібники	4. Призначена для користувача документація
5. Бізнес-плани	5. Технічна документація
6. Схеми організаційних структур	6. Копії екранів
7. Внутріофісну кореспонденція	
8. Протоколи нарад	

Вимоги, що формулюються у вигляді прецедентів (*use case requirements*), виявляються за допомогою вивчення існуючих організаційних документів, системних форм і звітів.

Прототипування (*prototyping*) – це найбільше часто використовуваний сучасний метод виявлення вимог. Програмні прототипи конструюються для візуалізації системи або її частини для замовників з метою одержання їхніх відгуків.

Існують два основні різновиди прототипів:

– *одноразовий прототип* («*throw-away*» *prototype*), що після того, як виявлення вимог завершене, просто відкидається. Розробка «одноразового» прототипу націлена тільки на етап

установлення вимог ЖЦ ПЗ. Як правило, цей прототип концентрується на найменш зрозумілих вимогах;

– *еволюційний прототип* (*evolutionary prototype*), що зберігається після виявлення вимог і використовується для створення кінцевого програмного продукту. Еволюційний прототип націлений на прискорення постачання продукту. Як правило, він концентрується на ясно викладених вимогах, так що першу версію продукту можна надати замовникові досить швидко (хоча її функціональні можливості, як правило, неповні).

JAD-метод повністю відповідає своїй назві – це спільна розробка додатків (*Joint Application Development*), здійснювана в ході одного або декількох нарад із залученням всіх учасників проєкту. Хоча ми відносимо *JAD*-підхід до сучасних методів виявлення вимог, цей метод був уперше уведений наприкінці 1970-х років компанією *IBM*.

JAD-метод ґрунтується на груповій динаміці. Групові зусилля більш перспективні з погляду одержання кращого вирішення проблем. Групи сприяють підвищенню продуктивності, швидше навчаються, схильні до більш кваліфікованих висновків, дозволяють виключити багато помилок, приймають ризиковані рішення, концентрують увагу учасників на найбільш важливих питаннях, об'єднують людей і т. д.

Метод швидкого розроблення додатків (*Rapid Application Development-RAD*) – це щось більше, ніж метод виявлення вимог – це цілісний підхід до розроблення ПЗ. Як ясно з назви методу, він припускає швидку поставку системних рішень. Технічна перевага відступає на друге місце порівняно зі швидкістю поставки.

Технологія *RAD* містить у собі п'ять підходів, перелічених нижче:

– еволюційне прототипування;

– *CASE*-засоби з можливостями генерації програм і циклічною розробкою з переходом від проєктних моделей до програми й назад;

– фахівці, що володіють розвиненими інструментальними засобами – *RAD* команда розробників. Кращі аналітики, проєктувальники й програмісти, які тільки може залучити організація. Команда працює в рамках строгого часового режиму та розміщується разом з користувачами;

– інтерактивний *JAD*-метод–*JAD*-сесія, під час якої секретарі замінюються бригадою *SWAT*, оснащеною *CASE*-засобами;

– жорсткі часові рамки (*timeboxing*) – метод керування проєктом, що відводить команді розробників фіксований період часу для завершення проєкту. Цей метод перешкоджає «розповзанню рамок проєкту»; якщо проєкт затягається, то рамки рішення звужуються, щоб дати можливість завершити проєкт вчасно.

2.2.2 Узгодження вимог

Вимоги, отримані від користувачів, можуть дублюватися або суперечити одне одному. Деякі вимоги можуть бути неясні або нереальні, інші вимоги можуть залишитися нез'ясованими. З цієї причини перш ніж вимоги потраплять до документу опису вимог, їх необхідно узгодити.

При умові що всі вимоги чітко ідентифіковані і пронумеровані можна сконструювати матрицю залежності вимог (матриця взаємодії).

Таблиця 2.2 Матриця залежності вимог

Вимоги	Т1	2	3	4
Т1	Х	Х	Х	Х
2	Конфлікт	Х	Х	Х
3			Х	Х

Продовження Таблиці 3.2 Матриця залежності вимог

Вимоги	Т1	2	3	4
4		Перекриття	Перекриття	Х

Суперечливі вимоги необхідно обговорити з замовниками і по можливості переформулювати, для пом'якшення протиріч (фіксацію протиріччя, видиму для подальшої розробки, необхідно зберегти).

Вимоги, що перекриваються, так само повинні бути сформульовані заново, що б виключити збіги.

Насправді узгодження й перевірка обґрунтованості вимог здійснюється паралельно з виявленням вимог. Після того як вимоги виявлені, вони піддаються певному рівню перевірки. Для всіх сучасних методів виявлення вимог, що пов’язані з так званою «груповою динамікою», це цілком природно. Як би там не було, після того як виявлені вимоги зібрані разом, вони в кожному разі повинні бути піддані ретельному обговоренню й перевірці.

Після того, як у результаті зняття протиріч і усунення повторів у вимогах, розроблено переглянутий набір вимог, їх необхідно піддати аналізу ризиків і призначити їм пріоритети. Аналіз ризиків спрямований на ідентифікацію вимог, що є потенційними джерелами труднощів у розробці. Призначення пріоритетів необхідне для того, щоб забезпечити можливість без труднощів змінити рамки проєкту у випадку виникнення непередбачених затримок.

Вимоги можуть бути «ризикованими» внаслідок впливу різних факторів. Вимогам властиві наступні типові види ризиків:

– технічний ризик, коли вимогу технічно важко реалізувати;

– ризик, зв’язаний зі зниженням продуктивності, коли вимога, будучи реалізованою, може несприятливо позначитися на часі реакції системи;

– ризик, пов’язаний з порушенням безпеки, коли вимога, будучи реалізованою, може створити пролом у захисті системи;

– ризик, пов’язаний з процесом розроблення, коли для реалізації вимоги необхідне використання незвичайних методів розроблення, незнайомих розроблювачам (наприклад методів формальної специфікації);

– ризик, пов’язаний з порушенням цілісності баз даних, коли вимога не може бути легко перевіреною та може призвести до суперечливості даних;

– політичний ризик, коли вимога може виявитися важкою для виконання із внутрішньополітичних причин;

– ризик, пов’язаний з порушенням законності, коли вимога може призвести до порушення чинних законів або очікуваної зміни закону;

– ризик, пов’язаний з мінливістю, коли вимога може потенційно змінюватися або еволюціонувати протягом процесу розроблення.

В ідеалі *пріоритети* вимогам призначають окремі замовники в процесі виявлення вимог. Потім вони узгоджуються на нарадах і знову змінюються після додавання до них факторів ризику.

2.2.3 Рівні вимог

Вимоги до ПО складаються з 3-х рівнів:

- бізнес вимоги;
- вимоги користувачів;
- функціональні вимоги.

До того ж, кожна система має свої нефункціональні вимоги.

Бізнес вимоги містять високорівневі цілі організації або замовників системи. Вони можуть бути записані в документі про спосіб і межах проєкту, в якому пояснюється, чому організації потрібна така система, тобто, описані цілі, які організація має

намір досягти з її допомогою. Вимоги користувачів описують цілі і завдання, які користувачам дозволить вирішити система. Ці вимоги можуть бути записані в документ про варіанти використання, де вказано, що клієнти зможуть зробити за допомогою системи.

Функціональні вимоги визначають функціональність системи, яку розробники повинні побудувати, щоб користувачі змогли виконати свої завдання в рамках бізнес вимог.

Системні вимоги – це високорівневі вимоги до продукту.

Бізнес правила включають корпоративні політики, урядова постанова, промислові стандарти і обчислювальний алгоритм.

Функціональні вимоги документуються в специфікації вимог до програмного забезпечення, де описується так повно, як необхідно очікуване поведінка системи. Специфікація вимог до ПЗ використовується при розробці, тестуванні, гарантії якості продукту, управлінні проєктом і пов'язаним з проєктом функцій.

На додаток до функціональним вимогам, специфікація містить нефункціональні, де описані цілі і атрибути якості.

Атрибути якості – це додатковий опис функцій продукту, виражені через опис його характеристик, важливих для користувачів або розробників (легкість і простота використання, легкість переміщення, цілісність, ефективність і стійкість до збоїв).

Обмеження стосуються вибору можливості розробки зовнішнього вигляду і структури проєкту.

Яких вимог не повинно бути:

- деталей дизайну або реалізації;
- даних про планування проєкту;
- відомостей про тестування.

Рисунок 2.1 – Область розробки технічних умов

У підетапи розробки вимог входять всі дії, що включають збір, оцінку та документування вимог, для програмного забезпечення або продуктів, що містять програмне забезпечення, в тому числі:

– ідентифікація класів користувачів для даного продукту;

– з'ясування потреб тих, хто представляє кожен клас користувачів;

– визначення завдань і цілей користувачів, а також бізнес-цілей, з якими ці завдання пов'язані;

– аналіз інформації, отриманої від користувачів, щоб відокремити завдання від функціональних і не функціональних вимог, бізнес-правил, передбачуваних рішень і надходять ззовні даних;

– розподіл низькорівневих вимог, за компонентами ПЗ, розподілених в системній архітектурі;

– встановлення відносної важливості атрибутів якості;

– встановлення пріоритетів реалізації;

– документування зібраної інформації і побудова моделей;

– перегляд специфікації вимог, який дозволяє впевнитися в тому, що запити користувачів усіма розуміються однаково, і усунення виникаючих проблем до передачі документа розробникам.

Управління вимогами – визначається як «вироблення і підтримку взаємної згоди з замовниками з приводу вимог щодо розроблюваного ПЗ». Ця угода втілюється в специфікації (у письмовій формі) і в моделях. Розробники також повинні прийняти задокументовані вимоги і висловитися за створення цього продукту. До дій з управління вимогами відносяться:

– визначення основної версії вимог (моментальний зріз вимог для конкретної версії продукту);

– перегляд передбачуваних змін вимог і оцінка ймовірності впливу кожної зміни до його прийняття;

– включення схвалених змін вимог до проєкту встановленими способами;

– узгодження плану проєкту з вимогами;

– обговорення нових зобов'язань, заснованих на оціненому вплив зміни вимог;

– відстеження окремих вимог до їх дизайну, вихідного коду та варіантів тестування;

– відстеження статусу вимог і дій зі зміни протягом усього проєкту.

2.2.4 Керування вимогами

Вимогами необхідно керувати. Керування вимогами являє собою частину загального керування проєктом. Воно пов’язане з трьома основними питаннями:

1. Ідентифікація, класифікація, організація й документування вимог.

2. Зміна вимог (за допомогою процесів, що встановлюють способи висування, узгодження, перевірки вірогідності та документування неминучих змін до вимог).

3. Простежуваність вимог (за допомогою процесів, що підтримують відносини взаємозалежності між вимогами й іншими системними артефактами, а також, власно, між вимогами).

Вимоги описуються природною мовою, наприклад: «Система повинна запланувати наступний телефонний дзвінок клієнтові по запиту», «Система повинна автоматично набирати запланований телефонний номер» і т. д.

Типова система може складатися з сотень або тисяч формулювань вимог. Для належного керування такою величезною кількістю вимог їх необхідно пронумерувати за допомогою певної *схеми ідентифікації.* Схема може включати *класифікацію* вимог у вигляді груп, що легше піддаються керуванню.

Існує декілька методів ідентифікації й класифікації вимог:

– *унікальний ідентифікатор* – звичайно послідовний номер, привласнений вручну або згенерований з використанням бази даних *case*-засобу;

– *послідовний номер усередині ієрархії документа* – привласнюється з урахуванням положення вимог у межах документа опису вимог;

– *послідовний номер у межах категорії вимог* – привласнюється на додаток до мнемонічного імені, що позначає категорію вимог.

Вимоги можна впорядкувати у вигляді ієрархічно впорядкованої структури, наприклад відношення батько-нащадок. Відношення батько-нащадок подібно відношенню композиції Батьківська вимога складається з дочірніх вимог. Дочірня вимога – це фактично «під-вимога» батьківської вимоги.

Ієрархічні відносини дозволяють увести додатковий рівень класифікації вимог. Це може безпосередньо позначатися в ідентифікаційному номері (вимога, пронумерована як 4.9, може бути дев'ятим нащадком «батька» з ідентифікаційним номером, рівним 4).

Простежуваність вимог (*requirements traceability*) – це всього лише частина керування змінами. Блок вимог технології керування змінами підтримує відносини простежуваності, щоб фіксувати

зміни, що виходять від або внесені у вимоги протягом ЖЦ розробки.

2.2.4 Бізнес-модель вимог

На етапі *встановлення вимог* здійснюється виявлення вимог і їх визначення, переважно у вигляді формулювань природною мовою. Формальне моделювання вимог з використанням мови UML проводиться пізніше на етапі аналізу або *специфікації вимог.* Проте під час установлення вимог постійно ведеться діяльність з узагальненого візуального подання зібраних вимог, що називається *бізнес-моделюванням вимог.*

Внаслідок того, що вимоги піддаються постійним змінам, напевно, найбільше занепокоєння при розробленні доставляє так зване *«розповзання рамок»* системи. Хоча деякі зміни вимог неминучі, необхідно суворо стежити за тим, щоб заявлені зміни не виходили за межі прийнятих рамок проєкту.

Щоб відповісти на запитання про рамки системи, необхідно знати, в якому контексті функціонує наша система. Необхідно знати, які зовнішні сутності – інші системи, організації, люди, машини й тощо – розраховують на отримання послуг від нас або готові надати послуги нам.

Тому рамки системи можна визначити, позначивши зовнішні сутності та вхідні/вихідні потоки даних між зовнішніми сутностями та нашою системою. Система, що проєктується, одержує вхідну інформацію та виконує необхідну обробку з метою вироблення вихідної інформації. Усяка вимога, що не може бути підтримана за рахунки внутрішньосистемних можливостей обробки, виходить за рамки системи.

Модель бізнес-прецедентів являє собою модель прецедентів на верхньому рівні абстракції. Модель бізнес-прецедентів визначає узагальнені бізнес-процеси. Бізнес-прецедент відповідає тому, що іноді називають можливостями системи. (Можливості

системи визначаються в документі, що описує бачення системи (system vision). Якщо при розробці системи наводиться документ опису бачення системи, він може використовуватись замість моделі бізнес-прецедентів).

Діаграма бізнес-прецедентів концентрується на архітектурі бізнес-процесів. Ця діаграма дає можливість глянути на передбачуване поводження системи так сказати «з висоти пташиного польоту». Неформальний опис кожного з бізнес-прецедентів повинний бути коротким, орієнтованим на ділову сторону системи, і концентруватися на основних потоках видів діяльності.

На етапі аналізу бізнес-прецеденти перетворюються в прецеденти. Саме на цьому етапі визначаються детальні прецеденти, і неформальний опис розширюється за рахунок включення в нього підпроцесів і альтернативних процесів, деяких копій екранів, що демонструють *GUI*-інтерфейс, а також взаємозв'язків між уведеними прецедентами.

Суб'єкти (actor) діаграми бізнес-прецедентів відрізняються від зовнішніх сутностей на діаграмі контексту. Суб'єкти активні. Вони керують процесом. Вони активізують прецеденти, відправляючи їм повідомлення про події. Прецеденти управляють подіями. Лінії, що зв'язують суб'єктів і прецеденти, – це не потоки даних. Ці лінії зв'язку являють собою потік подій, що виходять від суб'єктів і потік відгуків, що виходять від прецедентів.

Модель бізнес-класів – це модель класів. Як і у випадку з бізнес-прецедентами, різниця міститься в рівні абстрагування.

2.2.5 Документ опису вимог

Документ, що описує вимоги, є відчутним результатом етапу встановлення вимог

Шаблони для документів опису вимог широко доступні. Згодом кожна організація розробляє свої власні стандарти, які

відповідають прийнятій в організації практиці, корпоративній культурі й т. п.

Документ опису вимог повинен створити прецедент для системи.

Приклад змісту документа:

1. Попередні зауваження до проєкту
    a. Мета й рамки проєкта
    b. Діловий контекст
    c. Учасники проєкта
    d. Ідеї відносно рішень
    e. Огляд документа
2. Системні сервіси
    a. Рамки системи
    b. Функціональні вимоги
    c. Вимоги до даних
3. Системні обмеження
    a. Вимоги до інтерфейсу
    b. Вимоги до продуктивності
    c. Вимоги до безпеки
    d. Експлуатаційні вимоги
    e. Політичні і юридичні вимоги
    f. Інші обмеження
4. Проєктні питання
    a. Відкриті питання
    b. Попередній план-графік
    c. Попередній бюджет
5. Додатки
    a. Глосарій
    b. Ділові документи та форми
    c. Посилання

Основна частина документа опису вимог присвячена визначенню системних сервісів. Ця частина може займати до половини всього обсягу документа. Ця частина документа може містити узагальнені моделі – моделі бізнес-вимог.

Рамки системи можна моделювати за допомогою діаграми контексту.

Функціональні вимоги можна моделювати за допомогою діаграми бізнес-прецедентів. Однак діаграма охоплює перелік функціональних вимог тільки в найбільш загальному вигляді. Всі вимоги необхідно позначити, класифікувати й визначити.

Вимоги до даних можна моделювати за допомогою діаграми бізнес-класів.

Системні сервіси визначають, що повинна робити система. Системні обмеження визначають, наскільки система обмежена при виконанні обслуговування. Системні обмеження зв'язані з такими видами вимог:

– вимоги до інтерфейсу, що визначають як система взаємодіє з користувачами;

– вимоги до продуктивності, що у вузькому сенсі задають швидкість відгуку системи, з якої повинні виконуватися різні завдання;

– вимоги до безпеки, які описують права доступу користувача до інформації, контрольовані системою.

– експлуатаційні вимоги, які визначають програмно-технічне середовище, якщо воно відоме на етапі проєктування, у якому повинна функціонувати система.

– політичні й юридичні вимоги, які в основному мають на увазі.

2.3 Моделювання бізнес-процесів

Моделювання бізнес-процесів в останні роки стало актуальною тенденцією і використовується на практиці для вирішення широкого спектра завдань. Один з найбільш типових способів застосування подібних моделей – це вдосконалення самих модельованих процесів.

У разі, якщо діяльність є повторюваною, її називають процесом, у іншому випадку – проєктом.

Як правило, процеси становлять значну частину діяльності організації.

Процес – це пов'язаний набір повторюваних дій, які перетворять вихідний матеріал і (або) інформацію в кінцевий продукт або послугу відповідно до попередньо встановлених правил, враховуючи, що процес має кінцевий результат, розгляд діяльності компанії як сукупності процесів дозволяє більш оперативно реагувати на зміну зовнішніх умов, уникати дублювання діяльності та витрат, що не приводить до бажаного результату, і правильно мотивувати співробітників для його досягнення.

Моделювання бізнес-процесу зазвичай означає їх графічне формалізоване опис.

Побудова бізнес-моделі є одним з ключевих моментів специфікації вимог.

Бізнес-архітектура – це область, яка визначається вищими керівниками, відповідальними за основні функції організації і включає в себе твердження з приводу місій і цілей організації, критичні фактори успіху, бізнес стратегії, описи функцій а також структури і процеси, необхідні для реалізації функцій.

Ключем до побудови гарної бізнес-архітектури є визначення бізнес процесів, їх функцій і характеристик. Це стає основою для

побудови архітектури IT-додатків, які забезпечують автоматизовану підтримку цих процесів.

В рамках моделі бізнес-архітектури виділяються наступні основні компоненти:

– бізнес процеси / цілі і стратегія побудови бізнесу;
– організаційна компонента / організаційне оточення;
– інформація / інформаційне оточення;
– застосунок / оточення, що забезпечує.

Бізнес архітектура включає в себе наступні аспекти:

– бізнес стратегія, функції та організаційна структура – збори цільових установок, планів і структур організацій;

– архітектура бізнес-процесів, яка визначає основні функціональні області організації;

– показники результативності – цей аспект полягає у визначенні ключових показників результативності (КПР) роботи організації, їх поточних і бажаних рівнів, модель КПР використовується як засіб моніторингу виконання бізнес-процесів.

Balanced scorecard – широку популярність ця методика отримала, яка представляє собою систему, засновану на причинно-наслідкових зв'язках між стратегічними цілями, що відображають їх параметрами і факторами отримання планованих результатів. Вона розглядає 4 проєкції: фінансову, взаємини зі споживачем, операційні ефективності, мети і завдання яких взаємопов'язані і відображені фінансовими і нефінансовими показниками.

Balanced Scorecard складається з наступних питань:

– Financial - How do we appear to shareholders;
– Internal - At what processes should we excel;
– Innovation - What should we learn to grow and prosper;
– Customer - How do our Customers perceive us.

Структурно-організаційні компонента в моделі бізнес-архітектури відповідає на питання: хто за що відповідає в бізнес-процесах. Розподіл відповідальності за результати бізнес-процесу

визначаються у вигляді завдання ролей (повноважень), інкапсуляція даних ролей в бізнес процеси і закріплення ролей між конкретними персоналіями. Відповідно, організаційна компонента повинна підтримувати опис існуючої в організації організаційно-штатної структури, а також відображати закріплене в посадових інструкціях розподіл функціональних обов'язків учасників бізнес процесу. Метою розробки моделі організаційної компоненти є забезпечення можливості отримання якісних і кількісних оцінок, ефективності використання кадрових ресурсів в реалізації бізнес процесів, і як наслідок – пошук варіантів оптимізації організаційної структури підприємства.

Структура інформаційної компоненти: в рамках моделі бізнес архітектури зміст і детальність відображення інформаційної компоненти визначається ступенем її впливу на підтримку бізнесу. Інформаційна компонента повинна бути корелирующим відображенням бізнес архітектури. В рамках моделі бізнес-архітектури інформаційна компонента включає в себе всі ті інформаційні об'єкти (потоки, документи, дані), які безпосередньо пов'язані з бізнес подіями. Метою розробки моделі інформації та моделі даних є створення графічних уявлень потреб організації і окремих бізнес процесів в інформації. Це стає основою для реорганізації бізнес процесів і конструювання нових прикладних систем, опису взаємодій та інформаційного обміну, який відбувається між організацією і клієнтом-партнером. Інформаційна компонента представляється в такому вигляді, щоб була забезпечена можливість розгляду моделі інформації на різних рівнях розгляду абстракції, виходячи з потреб бізнес-процесу.

Організація компоненти «Застосунки» орієнтована на відображення того, які прикладні системи потрібні підприємству для виконання бізнес-процесів. Детальність опису прикладних систем повинна забезпечуватися на рівні, достатньому для розуміння складу автоматизуються функцій, які зберігаються (оброблюваних) операційних даних (документів)) що в кінцевому

підсумку дає об'єктивне уявлення про рівень її значущості для організації в цілому. Опис компоненти програми повинно бути не тільки досить для розуміння в якій частині бізнес процесів забезпечується підтримка, але і з точки зору оцінки витрат і вигод щодо використання системи.

2.3.1 Моделювання

Об'єктом моделювання може виступати будь-яка сутність, підходи по моделювання універсальні, і можуть бути застосовні як до архітектури корпоративно-операційної системи, або компанії в цілому, так і при проєктуванні окремих інформаційних систем.

Моделювання (по ISO-15704) – абстрактне уявлення реальності в будь-якій формі (наприклад, у фізичній, символічної, графічної або дескриптивної), призначене для подання певних аспектів цієї реальності і дозволяє відповідати на питання, що розглядаються.

Моделі можуть бути класифіковані за різними критеріями, наприклад:

1. Формальні (використовують загальноприйняті правила, нотації і засоби) і неформальні;

2. Кількісні (дозволяють виробляти чисельні оцінки і перевірки) і якісні (призначені для розуміння поведінки і структури системи);

3. Описові (призначені тільки для сприйняття людини) або виконувані (дозволяють досліджувати їх поведінку і використовувати отримані результати для висновків про вихідний об'єкті).

Загальні принципи моделювання:

1. Принцип здійсненності: створювана модель насамперед повинна забезпечувати досягнення поставлених цілей, таким чином перш ніж приступити до збору інформації про об'єкт

потрібно чітко визначити межі області моделювання, цілі і кількісні показники їх досягнення.

2. Принцип інформаційної достатності: при повній відсутності інформації про досліджуваний об'єкт побудова його моделі неможливо. При наявності повної інформації моделювання не має сенсу. Існує певний критичний рівень апріорних відомостей про об'єкт, при досягненні якого має сенс переходити від етапу збору інформації до етапу власне побудови моделі. В даному випадку закладаються умови для виконання такого значимого вимоги, як адекватність моделі, а саме досягнення розумного балансу між детальністю і споживчими якостями моделі.

3. Принцип множинності моделі: створювана модель повинна відображати ті властивості реального об'єкта, які впливають на обрані показники ефективності. При використанні будь-якої конкретної моделі пізнаються тільки деякі області дійсності. Для більш повного дослідження реального об'єкта необхідний ряд моделей, що дозволяють з різних сторін і з різною деталізацією відображати розглянутий процес.

4. Принцип агрегування: в більшості випадків складну систему можна представити у вигляді сукупності агрегатів (підсистем), для адекватного опису яких виявляються придатними деякі стандартні схеми.

5. Принцип відділення: досліджувана область як правило має в своєму складі кілька ізольованих компонентів, внутрішня структура яких досить прозора, або не подає безпосереднього інтересу для мети проєкту. В такому випадку її місце в моделі займає умовний порожній блок, для якого визначаються тільки значні вхідні і вихідні інформаційні потоки.

2.3.2 Об'єктний аналіз

Об'єктний аналіз – це метод дослідження не бізнес-процесів в цілому, а його неподільних найменших функціональних частин системи (на даному рівні розгляду) – структурних елементів (об'єктів), пов'язаних між собою деякими відносинами.

Процес:

– це безліч внутрішніх кроків діяльності, що починаються з одного і більше входу, і закінчуються створенням продукції, необхідної клієнту;

– це потік роботи, що проходить від одного фахівця до іншого або від одного відділу до іншого (в залежності від рівня розгляду);

– це процедура або набір процедур, які спільно реалізують бізнес-завдання або політичну мету підприємства, як правило в рамках організаційної структури, яка описує функціональні ролі і відносини;

– це взаємонезалежні компонент виробничої системи, що перетворює вхід в один або кілька виходів відповідно до попередньо встановлених правил;

– це пов'язаний набір повторюваних дій (функцій), які перетворять вихідний матеріал і / або інформацію в кінцевий продукт (послугу) відповідно до визначених критеріїв.

Процесний підхід до моделювання дозволяє:

1. Перейти від «точкового» текстового опису діяльності до повного формалізованого графічного опису діяльності, інтегруючим стрижнем якого є модельне уявлення бізнес-процесу.
2. Виділити і використовувати процеси як об'єкти управління
3. Змінити орієнтацію вектора управління компанії від «вертикальної» («на начальника») до «горизонтальної» («на замовника»).

2.3.3 Класифікація бізнес-процесів

Типи бізнес-процесів:

– основні (або ключові) процеси – стійкі процеси виробничо-господарської діяльності підприємства, орієнтовані на створення кінцевого продукту або послуги; в складі основних процесів може бути виділений контур керуючого впливу (власне дію і контроль за його виконанням).

– процеси, що забезпечують нормальне виконання основних процедур і змінюються в залежності від зміни складу, технологій основних процесів.

– процеси зовнішньої взаємодії – це процеси взаємодії з об'єктами, що не входять в узгоджене опис предметної області.

Компоненти процесу:

1) назва;
2) реалізована функція;
3) учасники;
4) відповідальна особа;
5) межі;
6) вхідні і вихідні потоки (вхідні потоки – це матеріали, послуги, та / або інформація, що перетворюються процесам для створення вихідними потоками. Вихідні потоки – це результат перетворення вхідних потоків);
7) необхідні ресурси – сприяючі чинники, які не перетворюються, щоб стати вихідним потоком (персонал, обладнання, приміщення, інформація);
8) мета процесу;
9) метрики процесу;
10) можливі ризики.

Власник процесу, що несе повну відповідальність за процес, і наділена повноваженнями щодо цього процесу. Він не стосується

функцій, які виконуються в рамках процесу окремими департаментами. Йому важлива успішна реалізація всього процесу, і перш за все його продуктивність, ефективність, адаптованість. Власник процесу забезпечує взаємодію з постачальниками вхідних потоків процесу і з споживачами його результатів.

Основні складові моделі бізнес-процесу – це функції, ресурси, документи і дані, учасники процесу, матеріали, продукти, послуги.

Для аналізу процесів рекомендується використовувати досвід консультантів, еталонні і референтні моделі, чек-листи, і інші статистичні методи, що застосовуються в сфері управління якістю.

Аспекти аналізу процесу:

– аналіз топології процесу;
– аналіз характеристик процесу;
– аналіз помилок процесу;
– аналіз динаміки виконання процесу;
– аналіз ризиків процесу;
– аналіз ресурсного оточення процесу;
– аналіз можливостей стандартизації процесу.

Аналіз топології процесу ставить собі за мету досягнути максимально зрозумілого перебігу процесу, що відображає при цьому або реальний стан речей, або оптимальний з урахуванням доступності ресурсів.

Етапи аналізу характеристик процесу:

1) визначення основних характеристик (показників процесу);
2) визначення метрик характеристик для їх оцінки;
3) моніторинг метрик характеристик процесу.

Основними характеристиками процесу є наступні показники:

– результативність – характеризує відповідність результатів процесу потребам і очікуванням споживачів;

– визначеність – відображає ступінь, з якою реальний процес відповідає опису;

– керованість – характеризує ступінь, в якій здійснюється управління виконання процесів виробництва необхідних продуктів або послуг, що відповідають певним цільовим показниками;

– ефективність – відображає, наскільки оптимально використовуються ресурси при досягненні необхідного результату процесу;

– повторення – характеризує здатність процесу створювати вихідні потоки з однаковими характеристиками при повторних його реалізаціях;

– гнучкість (адаптованість) – це здатність процесу пристосовуватися до зміни зовнішніх умов, перебудовуватися так, щоб це не призводило не результативна, не ефективність;

– вартість процесу – визначає сукупну вартість виконання функцій процесу і передачі результатів від однієї функції до іншої.

2.3.4 Етапи аналізу помилок процесу

Основні етапи:

– класифікація можливих помилок процесу;

– опис помилок процесу;

– виявлення помилок в процесі.

Можливі помилки, які можуть виникати при моделюванні бізнес-процесів:

– незавершеність – наявність прогалин в описі процесів;

– невідповідність – неадекватне використання інформаційних ресурсів в різних частинах процесу, що призводить до спотвореного сприйняття інформації або до неясності вказівок;

– ієрархічна несумісність – несумісність процесу з підпроцесами, його складовими;

– спадкова несумісність – наявність конфлікту між основними і подальшими процесами.

Динаміка процесів досліджується за допомогою динамічної (імітаційної) моделі.

Імітаційне моделювання – це методика, що дозволяє представляти в рамках динамічної комп'ютерної моделі протікання процесів, дії людей і застосування технологій, що використовуються в досліджуваних процесах.

2.3.5 Аналіз ризиків процесу

Операційний ризик – ризик прямих або непрямих збитків, який виникає в результаті невірного виконання бізнес-процесів, неефективності процедур внутрішнього контролю, технологічних збоїв, несанкціонованих дій персоналу або зовнішнього впливу.

Операційний ризик критичний для тих процесів, які характеризуються:

– значимістю для діяльності організації в цілому;
– великим числом транзакцій в одиницю часу;
– складністю системи технічної підтримки.

Етапи аналізу ризиків процесу:

1) структуризація ризиків;
2) опис ризиків та процесів, їх запобігання;
3) визначення ризиків в бізнес-процесах.

Аналіз ресурсного оточення процесу

Основу процесу складають виконувані функції, і для виконання кожної з них потрібно ресурси:

– людські – учасники процесу;
– виробничі;
– матеріальні;
– інформаційні;
– інтелектуальні – знання і повноваження учасників.

Аналіз можливостей стандартизації процесу (створення еталонних референтних моделей)

Еталони можуть бути базовими критеріями для інжинірингу бізнес-процесів. Складання власного бізнес процесу з аналогічним процесом, узятим за зразок, дозволяє отримати цільові чи орієнтовні показники. Така процедура називається еталонним порівнянням. Розбіжність між характеристиками еталонного процесу і власними показниками може підказати, як краще організувати у себе бізнес-проєкт.

2.3.6 Складові моделі об'єкта

Основні складові:

– методики;
– нотація;
– лінгвістичне забезпечення.
– Види нотації бізнес-процесів:
– IDEF0;
– BPMN 2.0 – Business Process Modeling Notation;
– UML.

Нотація BPMN – була розроблена в 2001 – 2004 рр групою BPMI.org, для стандартизованого опису бізнес процесів, зрозумілу як менеджерам і бізнес-аналітикам, так і розробникам ПО з можливістю подальшого збереження цього опису BPML. На відміну від UML, нотація BPMN включає лише ті елементи і поняття, які необхідні для моделювання бізнес-процесів. Її важливою особливістю є можливість встановити однозначну відповідність між однозначним описом елементів графічної нотації і описом мови на базі XML. У BPMN є тільки один тип діаграм – це діаграми бізнес-процесів (BPD), за допомогою яких описують послідовність виконання операцій в бізнес процесі.

Для побудови діаграми використовуються чотири види об'єктів:

1) потоку;

2) зв'язку;
3) розділові доріжки;
4) артефакти.

Дії зображуються прямокутниками з закругленими кутами. Вони підрозділяються на завдання (елементарні дії, які не підлягають декомпозиції) і підпроцеси (складові дії, які самі можуть бути представлені у вигляді бізнес-процесів). Підпроцеси можуть бути зображені на діаграмі в згорнутому або розгорнутому вигляді. Завдання, згорнуті і розгорнуті підпроцеси можуть бути обладнані маркерами, що вказують деякі характеристики їх виконання.

Маркери BPMN:

– маркер циклу;

– маркер багатопримірниковості;

– маркер компенсації.

Події BPMN служать для позначення різних подій, які можна почати, перервати і закінчити хід процесу. Проміжні і більшість початкових подій можуть бути забезпечені тригерами, які відображають суть події. Використання подій на діаграмах не є обов'язковим.

Шлюзи служать для управління розподілом і з'єднанням декількох ліній ходу процесу. Вони бувають єдиного, множинного, і складного вибору, а також паралельного виконання. Шлюзи єдиного вибору поділяються на засновані на даних (рішення про подальший перебіг процесу) і засновані на подіях; рішення приймається виходячи з того, що відбувається в цій точці події.

Шлюзи:

– оператор, що виключає АБО, керований даними;
– оператор, що виключає АБО, керований подіями;
– оператор, що включає АБО;
– оператор І.

2.3.7 Складний оператор

У BPMN визначено три типи зв'язків:

1) зв'язки потоку, що відображають послідовність виконання дій і з'єднує між собою об'єкти потоку (для них може бути задана умова переходу);

2) зв'язки повідомлень, що відображають потік повідомлень між учасниками бізнес-процесу;

3) асоціації, призначені для прив'язування до об'єкта потоку додаткової інформації у вигляді тексту або інших об'єктів.

До розділових доріжок відносяться пули і доріжки.

У вигляді пулів представляється учасник бізнес-процесу, наприклад, компанія, постачальник, клієнт і т.д. Пул може служити для поділу складових бізнес-процес дій між кількома учасниками, але може і не мати внутрішніх елементів, а представляти учасника процесу як чорний ящик. Якщо необхідно впорядкувати бізнес процес всередині пулу, його поділяють на доріжки, принцип розділення яких залишається на розгляд аналітика. Якщо бізнес-процес зображений всередині пулу, він не може виходити за його межі, тобто зв'язки потоку можуть перетинати кордони доріжок всередині пулу, але не межі пулу. Взаємодія поміщеного всередину пулу бізнес-процесу з зовнішнім світом моделюється за допомогою зв'язків повідомлень. Зв'язки повідомлень можуть починатися і закінчувати як на об'єктах потоку всередині пулу, так і на кордоні пулу, однак вони не повинні з'єднувати об'єкти всередині одного пулу.

2.4 Практичні завдання

Завдання 1. Розгляньте концепцію створення програмного забезпечення для інтернет-сайту лікарні. Дайте відповіді на такі запитання:

Для кого призначений цей додаток? Хто входить до кола зацікавлених осіб? Дайте оцінку потенційних покупців в Україні та за кордоном.

Укажіть три властивості, які повинна мати система, та три властивості, яких не повинно бути.

Вкажіть дві системи, з якими буде працювати ваша система.

Вкажіть три найбільш важливі ризики.

Завдання 2. Підготуйте реферат з методів, що використовуються при підготовці інтерв'ю та анкетування.

Завдання 3. Підготуйте порівняльну характеристику *RAD* та *JAD* технологій.

Завдання 4. Для проєкту системи «Чат-бот аптеки» сформулюйте типові види ризиків.

Завдання 5. Виконайте завдання 1 для системи клінічних досліджень.

Завдання 6. Виконайте завдання 1 для системи моніторингу стану здоров`я пацієнтів.

Завдання 7. Для системи «*Інтернет-магазин ліків»* сформулюйте 20 функціональних вимог, 10 вимог до якості продукту, 10 вимог до інтерфейсу, 5 вимог до апаратного забезпечення.

Завдання 8. Для системи «*Реєстратура лікарні*» сформулюйте 10 функціональних вимог з боку реєстратора, 10 функціональних вимог з точки зору хворого, 10 нефункціональних вимог.

Завдання 9. Розробіть прототипи інтерфейсу для системи «*Реєстратура лікарні*».

Завдання 10. Складіть анкету для виявлення вимог для системи «*Інтернет магазин ліків*».

Завдання 11. Підготуйте огляд методів що використовуються при керування вимогами.

Завдання 12. Розробіть документ Vision для системи «Інтернет-магазин *ліків*».

2.5 Контрольні запитання

1. Які існують способи пошуку концепції нової системи?

2. На які питання потрібна давати відповідь концепція системи?

3. Хто виконує виявлення вимог?

4. Які традиційні методи виконуються для виявлення вимог?

5. У чому головні вади та недоліки використання прототипування?

6. З якою метою використовується еволюційний прототип?

7. Які підходи об'єднує в собі *RAD*?

8. Як створюються ієрархії вимог?

9. Як виконується ідентифікація та класифікація вимог?

10. Які типові ризики властиві вимогам?

11. З якими питаннями пов'язано керування вимогами?

12. Чи існує єдиний стандарт документа опису вимог?

2.6 Література до розділу

1. Брагина, Т.И. Нечеткий анализ проєктного риска [Текст] / Т.И. Брагина, Г.В. Табунщик // Системи обробки інформації. Вип.3 (93). – 2011. – С. 15-21.

2. Вигерс, К. И. Разработка требований к программному обеспечению / К. И. Вигерс, Д. Битти. – СПБ: БХВ-Петербург, 2016. – 436 с.

3. Кон, М. Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ / Майк Кон. – М.:Издательский дом «Вильямс», 2012. – 256 с.

4. Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Роберт Мартин. – СПБ.:Издательский дом «Питер», 2011. – 464 с.

5. Пат. 81169 Україна, МПК (2011) МПК2012 G06Q 10/06, G06Q 10/10, G06Q 90/00. Спосіб керування ризиками проєктів [Текст] / Брагіна Тетяна Ігорівна; Табунщик Галина Володимирівна; заявник і патентовласник Запорізький національний технічний університет. – № u201214521; заявл. 18.12.2012; опубл. 25.06.2013, бюл. № 12.

6. Табунщик Г.В. Проектування, моделювання та аналіз інформаційних систем / Кудерметов Р.К., Притула А.В., Табунщик Г.В., Запоріжжя: ЗНТУ. -296 с.

7. Мацяшек Л. А. Анализ и проєктирование информа¬цион¬ных систем с помощью UML 2.0. ; пер. с англ. [Текст] / Мацяшек Л. А. – М.:Издательский дом «Вильямс», 2008. – 816 с

8. Хамбл, Д. Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ / Джез Хамбл. – М.:Издательский дом «Вильямс», 2011. – 436 с.

3 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

На етапі аналізу розробники займаються конструюванням моделей та намагаються досягти глибокого розуміння вимог.

Аналітична модель – це точне, чітке уявлення завдання, що дозволяє відповідати на запитання та конструювати рішення. Вона описує три аспекти об'єктів: їх статичну структуру (модель класів), взаємодії між ними (модель взаємодії) та життєві цикли об'єктів (модель станів).

Метою аналізу предметної області є розробка точної, чіткої, доступної для розуміння та конкретної моделі реального світу.

3.1 Модель класів предметної області

Модель предметної області відображає статичну структуру системи в реальному світі та ділить її на окремі елементи, зручні для оперування. Модель предметної області описує реальні класи та відносини між ними.

Конструювання моделі класів предметної області виконується у наведеній нижче послідовності:

– виділити класи;

– підготувати словник даних;

– виділити асоціації;

– виділити атрибути об'єктів і зв'язків;

–організувати та спростити класи за допомогою спадкування;

– перевірити наявність маршрутів для найбільш ймовірних запитів;

– перейти до наступної ітерації та вточнити модель;

– переглянути рівень абстрагування;

– згрупувати класи в пакети.

Виявлення класів

Існує безліч підходів до виділення класів. Найбільш поширені:

–підхід на основі використання іменних груп;

–підхід на основі використання загальних шаблонів для класів;

–підхід на основі використання прецедентів;

–підхід CRC (Class-Responsibility-Collaboration).

Підхід на основі використання іменних груп (тобто іменників) припускає, що аналітик читає формулювання *документа опису вимог* у пошуках іменних груп. Кожний іменник розглядається як потенційний клас. Потім список усіх класів поділяється на наступні три групи:

– релевантні, або відповідні класи;

– нечіткі, або сумнівні класи;

– нерелевантні, або невідповідні класи.

До *нерелевантних* належать класи, що виходять за рамки проблемної області. Для них не вдається дати формулювання їх призначення.

До *релевантних* класів відносяться класи, що належать до проблемної області. Іменники, що відображають імена цих класів, часто зустрічаються в документі опису вимог.

До *нечітких* належать класи, що не можна впевнено і беззастережно визнати відповідними. Їх аналізують більш глибоко, апотім відносять до однієї з попередніх груп.

Підхід на основі використання загальних шаблонів для класів дозволяє вивести потенційні класи на основі теорії родової класифікації об’єктів.

Барамі пропонує такий перелік груп:

– понятійний (концептуальний) клас. Він являє собою ідею, яку поділяє або з якою згодна значна спільність людей. Наприклад, клас «*Резервування*»;

– клас подій. Подія не вимагає часу стосовно тимчасової шкали, що розглядається. Наприклад, клас «*Прибуття*»;

– організаційний клас. Організація – це будь-який вид цілеспрямованого об'єднання або сукупності сутностей. Наприклад, клас «*Туристичне бюро*»;

– клас людей. Під людьми тут розуміється роль, яку людина грає в тій чи іншій системі. Наприклад, клас «*Пасажир*»;

– клас місцерозташувань. Наприклад, клас «*Офіс*».

Дж. Рамбау, А. Джекобсон та Г. Буч пропонують наступну схему класифікації:

– фізичний клас (наприклад, клас «*Літак*»);

– бізнес-клас (наприклад, клас «*Резервація*»);

– логічний клас (наприклад, клас «*Розклад*»);

– прикладний клас (наприклад, клас «*Операція резервування*»);

– комп'ютерний клас (наприклад, клас «*Індекс*»);

– поведінковий клас (наприклад, клас «*Скасування резервування*»).

Даний підхід служить скоріше як корисне керівництво, але не визначає систематичного процесу, за допомогою якого можна було б виділити надійну та повну множину класів.

Підхід на основі використання прецедентів. Цьому підходу надається особливе значення в мові *UML*. Можна навіть сказати, що цей підхід рекомендується використовувати в рамках методології *RUP* (*Rational Unified Process*). Графічна модель прецедентів супроводжується неформальними описами, а також діаграммами послідовностей і кооперації для окремих прецедентів. Ці додаткові описи та кроки визначення діаграм (і об'єктів) потрібно виконати для кожного прецеденту. На основі

цієї інформації можна прийти до узагальнень, необхідних для виявлення потенційних класів.

Підхід, що керується прецедентами, має особливості, притаманні підходу знизу-вгору. Після того, як прецеденти стають відомі, а *уявлення* про систему *з точки зору взаємодії,* щонайменше, частково визначено за допомогою діаграм послідовностей, об'єкти, що використовуються в цих діаграмах, приводять до виявлення класів.

Насправді цей підхід у чомусь схожий на підхід, який використовує іменні групи. Їх об'єднує те, що прецеденти специфікують вимоги. Обидва підходи спрямовані на вивчення формулювань, викладених у документі опису вимог, щоб виявити в підсумку потенційні класи. Те, що ці формулювання викладаються в оповідній формі або подані графічно, має другорядне значення. У будь-якому випадку на цьому етапі ЖЦ розробки ПЗ більшу частину прецедентів можна описати лише в текстовій формі без діаграм взаємодії.

Підхід, заснований на прецедентах, має ті ж недоліки, що йпідхід, який використовує іменні групи. Будучи по суті підходом знизу-вгору в сенсі точності, він спирається на повноту та коректність моделей прецедентів. У результаті, він навіть може призвести до небажаного розбалансування ітеративного інарощуваного процесу розроблення ПЗ, при якому моделі прецедентів повинні бути завершені ще до побудови моделей класів. Загалом, які б не були цілі та засоби, це призводить до *функціонального підходу* (*function-driven approach*), прихильники об'єктно-орієнтованого підходу вважають за краще називати його *проблемно-орієнтованим (problem-driven).*

Підхід CRC. Підхід *CRC* (*Class – Responsibility – Collaborators* – клас – відповідальність – «співробітники») являє собою щось більше, ніж метод виявлення класів, – це привабливий спосіб інтерпретації та вивчення об'єктів (а також і навчання об'єктному підходу). Найбільшу популярність підхід *CRC* отримав завдяки

роботам Ребеки Вірфс-Брок (*Rebecca Wirfs-Brock*) та її колег Б. Вілкерсон (*B. Wilkerson*) і Л. Вінер (*L. Wiener*).

Підхід *CRC* включає в себе сеанси «мозкового штурму», проведення яких полегшується за рахунок використання спеціально підготовлених карток. Картки складаються з трьох відділень: *ім'я класу* записується у верхньому відділенні, *обов'язки класу* перераховані в лівому відділенні, а *співробітники* перераховані в правому відділенні. Обов'язки – це послуги (операції), які клас готовий виконати в інтересах інших класів. Для виконання багатьох обов'язків необхідна участь (обслуговування) з боку інших класів. Такі класи перераховуються як «співробітники».

Процес *CRC* – це живий процес, під час якого розробники «грають в карти», вони заповнюють картки іменами класів і призначають їм «обов'язки» та «співробітників» у ході виконання сценарію оброблення інформації (наприклад, сценарію прецеденту). У тих випадках, коли виникає потреба в якійсь послузі, аіснуючі класи не покривають її, створюється новий клас, якому призначаються відповідні «обов'язки» та «співробітники». Якщо клас стає «надто зайнятим», він розділяється на кілька менших класів.

Підхід *CRC* відрізняється від інших підходів тим, що при його використанні виділення класів є результатом аналізу повідомлень, переданих між об'єктами для виконання завдань оброблення інформації. Акцент робиться на уніфікованому методі розподілу «інтелекту» в системі, і деякі класи можуть бути швидше отримані, виходячи з подібної технічної потреби, ніж виявлені у якості «бізнес-об'єктів» як таких. У цьому сенсі метод *CRC* може бути більш прийнятним для перевірки правильності вибору класів вже виявлених з допомогою інших методів. Підхід *CRC* також корисний при встановленні властивостей класів (що логічно випливають із «обов'язків» і типів «співробітників» класу).

Підготовка словника даних

Слова допускають дуже багато інтерпретацій, тому для всіх елементів моделі необхідно підготувати словник даних. Для кожного класу слід придумати опис розміром в один абзац. Треба описати область застосування класу в рамках даної задачі, вказати всі припущення та обмеження, що стосуються його використання. Словник даних повинен містити опис асоціацій атрибутів, операцій і значень перерахованих типів.

Наприклад словник даних для системи підтримки банкоматів може містити наступні терміни:

Рахунок – окремий рахунок в банку, з яким виконуються операції. Рахунки можуть бути різних типів. Клієнт може мати декілька рахунків.

Банкомат – термінал, що дозволяє клієнту виконувати операції, використовуючи для ідентифікації кредитні картки. Банкомат взаємодіє з клієнтом, отримуючи від нього дані, відправляючи інформацію про операцію на центральний комп'ютер для її перевірки та обробки, також видає клієнту готівку.

Банк – фінансова установа, що зберігає рахунки клієнтів та видає кредитні картки для доступу до рахунків за допомогою банкоматів.

Клієнт – власник одного або декількох рахунків у банку. Клієнт – це не обов'язково одна особа, це може бути декілька осіб або організація. Одну і ту ж людину, що має рахунок у декількох банках, слід розглядати як декілька клієнтів.

Виділення асоціацій

Структурне відношення між двома та більше класами єасоціацією. Асоціації часто відповідають дієсловам стану або дієслівним групам. До них належать характеристики фізичного розміщення, спрямовані дії, передання інформації та виконання будь-яких умов.

При виділенні асоціації між класами найбільш стандартними є такі:

– клас *A* є фізичною частиною класу *B* («*Крило*» – «*Літак*»);

– клас *A* є логічною частиною класу *B* («*Відрізок дороги*» – «*Маршрут польоту*»);

– клас *A* фізично міститься в класі *B* («*Пасажир*» – «*Літак*»);

– клас *A* логічно міститься в класі *B* («*Політ*» – «*Графік польоту*»);

– клас *A* є описом класу B («*Опис польоту*» – «*Політ*»);

– клас *A* є елементом транзакції класу *B* («*Послуга*» – «*Журнал технічного обслуговування*»);

– клас *A* відомий (записаний, включений) у клас *B* («*Замовлення квитка*» – «*Декларація*»);

– клас A є організаційною одиницею класу *B* («*Служба підтримки*»– «*Літак*»);

– клас *A* використовує або керує класом *B* («*Пілот*» – «*Літак*»);

– клас *A* пов'язаний з транзакцією класу *B* («*Пасажир*» – «*Квиток*»).

Потім необхідно видалити зайві асоціації, для чого можна користуватися такими рекомендаціями:

1. Асоціації між класами, що були видалені на попередніх етапах. Якщо один із класів, зв'язаних асоціацією, був видалений, асоціацію теж потрібно видалити або переформулювати в термінах інших класів.

2. Несуттєві, або асоціації, що відносяться до реалізації. Викиньте всі асоціації, що лежать за межами області завдання або ті, що описують конструкції, які відносяться до реалізації.

3. Дії. Асоціація повинна описувати структурну властивість області додатка, а не короткочасну подію. У деяких випадках вимога виражається у вигляді дії, проте має на увазі деяке структурне відношення. Таке твердження потрібно переформулювати.

Приклад з банкоматом. Асоціація *«Банкомат приймає банківську карту»* описує частину циклу взаємодії банкомата з

клієнтом, а не постійне відношення банкомата та карти. З тієї ж причини можна виключити асоціацію *«Банкомат взаємодіє з користувачем»*.

4. Тернарні асоціації. Більшість *n*-арних асоціацій можна виразити через бінарні, додавши відповідні кваліфікатори. Якщо термін у тернарній асоціації є описовим і не має власної індивідуальності, він є атрибутом бінарної асоціації. Наприклад, асоціацію «*Компанія виплачує співробітнику зарплату*» можна переформулювати у вигляді бінарної асоціації «*Компанія наймає співробітника*», причому кожен зв'язок «*Компанія-Співробітник*» буде характеризуватися своїм значенням *salary* (зарплата).

Іноді в додатку дійсно потрібна тернарна асоціація. Наприклад, структуру *«Викладач читає курс лекцій в аудиторії»* не можна розбити на бінарні асоціації без втрати інформації.

5. Похідні асоціації. Відкиньте асоціації, що можуть бути виражені через інші асоціації. Наприклад, *GrandparentOf (Дідусь)* можна виразити через пару асоціацій *ParentOf (Родитель)*. Викидайте й ті асоціації, що виражаються як обмеження на атрибути Наприклад, *youngerThan (молодший)* виражає умову, що стосується дат народження двох людей, а не якусь додаткову інформацію.

Класи, атрибути та асоціації моделі класів повинні відображати максимально незалежну інформацію. Наявність безлічі маршрутів між класами часто вказує на похідні асоціації, що можуть бути виражені через деякі примітивні асоціації.

Не всі асоціації, що утворюють множинні маршрути між класами, є надлишковими. Іноді існування асоціації можна вивести з декількох примітивних асоціацій, а от її кратність таким шляхом визначити не можна. У подібній ситуації асоціацію слід зберегти, при умові, що обмеження на кратність важливо для моделі. Наприклад, на рис. 3.1 компанія наймає безліч співробітників і володіє безліччю комп'ютерів. Кожному співробітнику надається нуль і більше комп'ютерів для

персонального використання. Деякі комп'ютери призначені для загального користування і не приписані до жодного співробітника. Кратність асоціації «*Приписаний до*» не може бути виведена з кратності асоціацій «*Наймає*» та «*Володіє*».

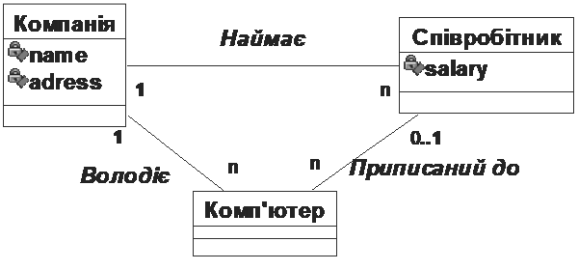


Рисунок 3.1 – Приклад виділення асоціацій

Хоча похідні асоціації не вносять додаткової інформації до моделі, вони можуть бути корисні в реальному світі та при проєктуванні Наприклад, відношення родинності типу «дядя», «теща» та «кузен» мають власні назви, тому що вони описують типові відношення, що вважаються досить важливими в нашому суспільстві. Якщо відносини такого роду особливо важливі для моделі, їх можна залишити на діаграмі класів, вказавши перед ім'ям символ навскісної риски (/), що означає залежний статус та дозволяє відрізняти їх від фундаментальних асоціацій.

Далі слід проаналізувати семантику асоціацій:

– назва повинна говорити не про те, як або чому виникла певна ситуація, а про те, в чому ця ситуація полягає. Назви важливі для розуміння моделі в цілому, а тому вибирати їх слід дуже обережно. Наприклад, «*Банківський комп'ютер обслуговує рахунки*» – це твердження, що описує дію. Назву асоціації правильніше буде переформулювати як «*Банк керує рахунком*»;

– назви полюсів асоціацій потрібно вказувати скрізь, де вони мають сенс. Наприклад, в асоціації «*Працює на*» клас «*Компанія»* відносно класу «*Людина*» є роботодавцем, а «*Людина*» відносно

класу «*Компанія*» являє собою співробітника. Асоціація між двома примірниками одного і того ж класу вимагає наявності імен полюсів, за допомогою яких можна було б відрізняти ці екземпляри. Наприклад, в асоціації «*Людина керує Людиною*» полюси будуть називатися «*начальник*» та «*підлеглий*»;

– використовуйте кваліфіковані асоціації. Зазвичай назва ідентифікує об'єкт у рамках певного контексту. Більшість назв не є унікальними в масштабах всієї системи (глобально). Контекст разом з ім'ям дозволяють унікально ідентифікувати об'єкт. Кваліфікатор дозволяє відрізняти один від одного об'єкти, що перебувають на полюсі асоціації з кратністю «багато»;

– необхідно завжди вказувати кратність асоціації, але не намагайтеся визначити її точно на першому етапі моделювання. Кратність часто змінюється в процесі аналізу. Перевіряйте асоціації з кратністю «один». Наприклад, асоціація «*Один Менеджер керує безліччю Працівників*» не дозволить створити матричну систему управління або описати співробітника, відповідального перед кількома начальниками. Подумайте про необхідність введення кваліфікаторів для асоціацій з кратністю «багато», а також про упорядкування об'єктів;

– додайте всі пропущені асоціації, що вам вдасться виявити;

– агрегація важлива для деяких видів додатків, зокрема для опису деталей механізмів і специфікацій матеріалів. Для інших додатків важливість агрегації не настільки велика, і не завжди буває зрозуміло, чи слід її використовувати замість звичайної асоціації. Не витрачайте занадто багато часу на визначення типу асоціації. Виберіть те, що відразу здається вам більш правильним, і рухайтеся далі.

Наприклад будемо вважати «*Банк*» частиною «*Консорціуму*» іпозначимо відношення між ними як агрегацію.

Приклад з банкоматом. На рис. 3.2 наведена діаграма класів з нанесеними на неї асоціаціями. На ній вказані імена тільки для найбільш важливих асоціацій. На діаграмі зазначені значення

кратності. Деякі рішення були довільними. Не варто турбуватися про це: існує безліч коректних моделей завдання.

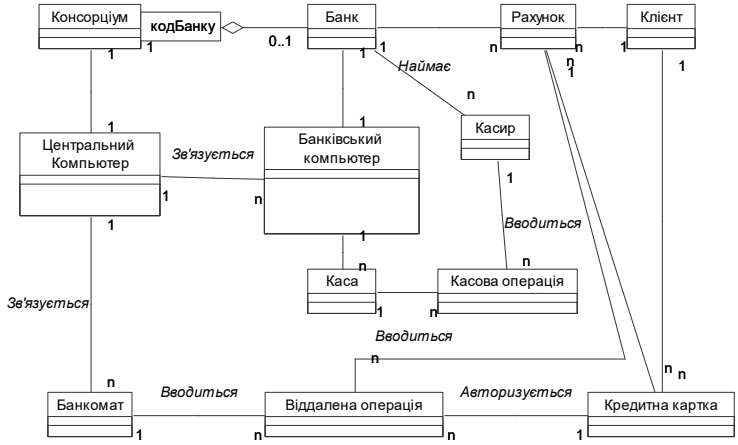


Рисунок 3.2 – Вихідна діаграма класів для банкомата

Виділення атрибутів.

Атрибути – це властивості об'єктів, такі як вага, швидкість або колір. Значення атрибутів не повинні бути об'єктами; щоб показати відношення між об'єктами, слід використовувати асоціації.

Атрибути зазвичай наявні в описі завдання у вигляді іменників, що беруть участь в присвійних зворотах, на зразок «колір машини» або «розташування курсора». Прикметники часто відповідають конкретним значенням атрибутів-перерахувань (наприклад, «червоний», «включено», «застарілий»). На відміну від класів і асоціацій атрибути навряд чи будуть повністю перераховані в описі завдання. Щоб виділити їх, усі доведеться спиратися на своє знання області завдання та реального світу. Атрибути також присутні в артефактах споріднених систем

При виділенні атрибутів треба розглядати тільки ті з них, що мають безпосереднє відношення до додатка. Спочатку займіться найбільш важливими атрибутами, дрібні деталі можна буде додати в модель пізніше. У процесі аналізу не витрачайте час на деталі реалізації. Обов'язково треба давати всім атрибутам значущі імена.

Зазвичай похідні атрибути включати в модель на цьому етапі не слід. Наприклад, вік можна визначити за датою народження і поточною датою (остання є властивістю оточення).

Треба також шукати атрибути і для асоціацій. Ці атрибути є властивостями зв'язків між об'єктами, а не властивостями індивідуальних об'єктів. Наприклад, асоціація «багато-до-багатьох» між «*ОдержувачАкцій*» і «*Компанія*» має атрибут «*кількість Акцій*».

Треба виключити непотрібні та некоректні атрибути, коли вони:

– об'єкти. Якщо важливою рисою елемента є незалежне існування, то цей елемент – об'єкт, а не атрибут. Наприклад, «*Начальник*» – це клас, а «*зарплата*» – атрибут. Різниця часто залежить від додатка. Елемент, що володіє власними рисами в рамках даного додатка, має моделюватися як клас;

– кваліфікатори. Якщо значення атрибуту залежить від конкретного контексту, його можна спробувати переформулювати у вигляді кваліфікаторів. Наприклад, «*номер Співробітника*»– це не унікальна характеристика людини, зайнятої на двох роботах, це, скоріше, кваліфікатор асоціації «*Компанія наймає співробітника*»;

– імена. Імена краще моделювати як кваліфікатори, а не як атрибути. Дайте відповіді на наступні запитання: чи служить ім'я для вибору унікального об'єкта з множини? Чи може об'єкт, що належить множині, мати більше одного імені? Якщо відповідь позитивна, ім'я служить кваліфікатором асоціації. Якщо ім'я здається унікальним, можливо, ви забули додати в модель клас, для якого воно служить кваліфікатором.

Наприклад, «*назва Відділу*» може бути унікальною в рамках компанії, але коли-небудь вам доведеться працювати з декількома компаніями. Краще відразу використовувати відповідну кваліфіковану асоціацію. Ім'я є атрибутом у тому випадку, якщо його використання не залежить від контексту, особливо якщо воно не унікальне в деякій множині. Імена людей, на відміну від назв компаній, можуть повторюватися, а тому є атрибутами:

– ідентифікатори. Об'єктно-орієнтовані мови використовують поняття ідентифікатора об'єкта для позначення однозначного посилання на об'єкт. Не слід включати в модель атрибут, єдиним призначенням якого є ідентифікація, тому що ідентифікатори присутні в моделях класів неявним чином. Перераховуйте тільки ті атрибути, що присутні в області додатка. Наприклад, кодРахунку – це справжній атрибут, тому що банк призначає кожному рахунку свій код, а клієнт бачить цей код. Навпаки, внутрішній «*ідентифікатор Транзакції*» не є атрибутом, хоча на етапі реалізації його використання може бути вигідним;

– атрибути асоціацій. Якщо існування значення вимагає існування зв'язку, відповідна властивість є атрибутом асоціації, ане одного зі зв'язаних нею класів. Атрибути зазвичай досить ясно виділяються з асоціацій «багато-до-багатьох»: їх не можна прикріпити до жодного з класів через їх кратності. Наприклад, атрибут «*дата Вступу*» належить асоціації між «*Людиною*» і «*Клубом*», тому що людина може належати до кількох клубів, а клуб може мати кількох членів. Асоціації «один-до-багатьох» деталізувати складніше, тому що тут можна прикріпити атрибут до одного з класів без втрати інформації. Чиніть опір цьому бажанню, тому що якщо кратність зміниться, модель стане некоректною. Ті самі проблеми виникають і з асоціаціями типу «один-до-одного».

– внутрішні значення. Якщо атрибут описує внутрішній стан об'єкта, невидимий зовні, його слід виключити з аналітичної моделі.

– зайві деталі. Виключіть незначні атрибути, які не впливають на більшість операцій.

– нетипові атрибути. Атрибут, що повністю відрізняється від усіх інших і не зв'язаний з ними, може вказувати на те, що клас, до якого він належить, слід розділити на два різні класи. Клас повинен бути простим і цільним. Змішування різних класів – одна з основних причин появи ненадійних моделей. Нечітко визначені класи часто бувають результатом занадто раннього прийняття рішень, що стосуються реалізації.

– логічні атрибути. Розгляньте всі логічні атрибути ще раз. Часто логічний атрибут можна розширити та переформулювати у вигляді перерахування.

Реструктурування за допомогою спадкування.

Наступний крок: організація класів за допомогою спадкування шляхом виявлення їх загальної структури. Спадкування може бути використано двома способами: як узагальнення однакових аспектів існуючих класів у суперкласах (знизу вгору) і як конкретизація існуючих класів безліччю підкласів (зверху вниз).

Узагальнення знизу вверх. Спадкування можна простежувати знизу вгору шляхом пошуку класів з однаковими атрибутами, асоціаціями та операціями. Для кожного узагальнення слід визначити суперклас, в якому міститимуться загальні риси. Для цього вам, можливо, доведеться перевизначити деякі атрибути або класи. Деякі узагальнення копіюються з реального світу. Скрізь, де це можливо, слід використовувати існуючі поняття. Відсутні класи можна виписувати з міркувань симетрії.

Наприклад, класи «*Віддалена операція*» та «*Касова операція*» подібні одне одному (за винятком ініціалізації) і можуть бути узагальнені класом «*Операція*».

Конкретизація зверху вниз. Конкретизація звичайно слідує з опису області додатка. Шукайте іменні групи, що складаються зрізних прикметників з ім'ям класу: фіксоване меню, меню, що

розкривається та меню, що висувається. Уникайте надмірного уточнення. Якщо запропонована конкретизація несумісна з існуючим класом, можливо, цей клас просто неправильно сформульований.

Узагальнення та перерахування. Перераховані підкласи області додатка найчастіше потрапляють під визначення конкретизації. Зазвичай буває досить відзначити існування безлічі перерахованих окремих випадків без явної їх вказівки. Наприклад, «*Рахунок»* (у прикладі з банкоматом) може бути рахунком до запитання або ощадним рахунком. У деяких банківських застосуваннях такий розподіл може бути дуже важливим, проте на поведінку банкомата він не впливає, тому «*тип рахунка»* можна зробити просто атрибутом класу «*Рахунок*».

Множинне спадкування. Його краще використовувати лише тоді, коли це дійсно необхідно, оскільки воно підвищує концептуальну і технічну складність моделі.

Подібні асоціації. Якщо назва асоціації з'являється в моделі кілька разів, причому несе вона при цьому однаковий сенс, спробуйте узагальнити асоційовані класи. Інколи в таких класів може не бути нічого спільного, за винятком асоціації, але досить часто ви виявлятимете загальні риси, пропущені на попередніх етапах.

Наприклад, «*Операція*» вводиться як за допомогою класу «*Каса*», так і в класі «*Банкомат*». Клас «*Пристрій Вводу*» узагальнює класи «*Каса*» та «*Банкомат*».

Коректування рівня спадкування. Атрибути й асоціації мають бути присвоєнні конкретним класам з ієрархії. Кожен з них має бути присвоєний найбільш загальному класу, до якого він застосовний. Для цього вам можуть знадобитися деякі коректування. З симетрії може виходити наявність додаткових атрибутів, які дозволять чіткіше відрізняти підкласи один від одного.

На рис. 3.3 показана модель класів банкомата після додавання спадкування.

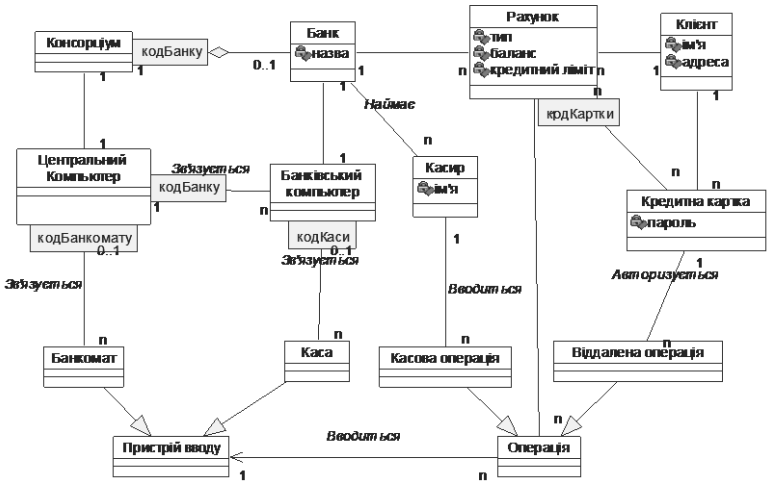


Рисунок 3.3 – Перетворена модель класів банкомата

Перевірка маршрутів

При перевірці маршрутів у моделі необхідно отримати відповіді на такі запитання: Якщо в певному місці передбачається унікальне значення, чи існує маршрут, що дає унікальний результат? Чи передбачений метод вибору унікальних значень об'єктів, охарактеризованих кратністю «багато»? Чи є важливі питання, на які ви не можете відповісти?

Питання без відповіді вказують на інформацію, якої бракує. Якщо щось просте з реального світу в моделі виявилося складним, це означає, що пропущена важлива інформація (проте обов'язково перевірте, чи не властива та ж складність реальному світу).

У моделі можуть бути присутні класи, не зв'язані з іншими. Зазвичай це відбувається тоді, коли відношення між такими класами і останньою моделлю носить розмитий характер. Проте завжди перевіряйте такі класи, тому що ви могли просто пропустити існуючу асоціацію.

Ітераційне розроблення моделі класів

Модель класів рідко буває правильною після першого проходу. Весь процес розроблення ПЗ будується на ітераціях. Різні частини моделі звичайно знаходяться на різних стадіях розробки. Якщо був знайдений недолік, необхідно повернутись на попередню стадію та усунути його. Деякі уточнення можуть бути зроблені тільки після розроблення моделі станів та взаємодії.

Групування класів у пакети

Останній етап моделювання – групування класів у пакети.

Пакет – це група елементів (класів, асоціацій, узагальнень, вкладених пакетів), що характеризуються загальною темою. Пакети роблять модель більш зручною з точки зору конструювання, друку та огляду. Класи одного пакету зв'язані друг з другом міцніше, ніж класи в різних пакетах.

Для того щоб розподілити класи між пакетами потрібно знайти точку з'єднання – класи, що єднають частини моделей, які не мають інших зв'язків. Треба обирати пакети так, щоб зменшувати кількість перетинань на діаграмах класів.

3.2 Модель станів предметної області

Деякі об'єкти предметної області за час свого існування змінюють декілька якісно різних станів. У цих станах вони можуть мати різні обмеження та значення атрибутів, різні асоціації або кратності, виконувати різні операції або мати різну поведінку і т. д. Для таких класів корисно побудувати діаграму станів. Діаграма станів описує різні стани, в яких може знаходитися об'єкт, властивості об'єкта та діючі на них обмеження, а також події, що викликають перехід об'єкта з одного стану в інший.

Більшість класів предметної області не вимагають використання діаграм станів. Для їх опису досить списку операцій.

Модель станів може допомогти в розумінні поведінки тих класів, що можуть перебувати у суттєво різних станах.

Спочатку потрібно виявити класи, які можуть перебувати в різних станах, і записати стан для кожного класу. Потім необхідно визначити події, що викликають перехід кожного об'єкта з одного стану в інший. Знаючи стан і події, ви можете побудувати діаграму станів для кожного з об'єктів. Нарешті, ви повинні перевірити отримані діаграми на повноту і коректність.

Модель станів предметної області конструюється в декілька етапів:

– виявлення класів, що мають різні стани;
– виділення станів;
– виділення подій;
– побудова діаграм станів;
– перевірка діаграм станів.

Виявлення класів з різними станами

На цьому етапі потрібно вивчити перелік класів предметної області, що чітко характеризуються певним життєвим циклом. Треба відшукайте класи, що розвиваються або мають циклічну поведінку, та ідентифікувати значимі стани в життєвому циклі кожного з об'єктів. Наприклад, «*Наукова стаття*» для певного журналу послідовно переходить зі стану ***написання*** в стан ***рецензування***, а потім або в стан ***прийнята,*** або ***відкинута***. Стани статті можуть змінюватися циклічно, наприклад, якщо рецензенти вимагатимуть внесення змін або доповнень, але основна послідовність станів може бути охарактеризована як розвиток.

Деякі стани можуть бути присутніми не в кожному циклі; можуть існувати стани і поза циклом. Бувають класи і з хаотичним життєвим циклом, але більшість класів можуть бути віднесені або до тих, що розвиваються, або до циклічних.

Наприклад, «*Рахунок*» – це важливе поняття з області бізнесу. Поведінка банкомата залежить від стану рахунку. Життєвий цикл

рахунку – суміш послідовної та циклічної поведінки. Інші класи предметної області банкомата не мають значимих моделей стану предметної області.

Виділення станів

Наступний крок – треба перерахувати стани для кожного з класів. Потрібно охарактеризувати об'єкти кожного класу: вказати значення атрибутів, що може мати об'єкт; асоціації, в яких він може брати участь; значення кратності вузлів цих асоціацій; визначити атрибути та асоціації, що мають сенс тільки в певних станах. Надати кожному стану осмислену назву. Назва не повинна описувати, яким чином стан було отримано, вона має позначати сам стан. Стани повинні виділятися на підставі якісних відмінностей поведінки, атрибутів або асоціацій.

Не обов'язково визначати всі стани до виділення подій. Розглянувши події та переходи між станами, можна буде знайти відсутні стани.

Наприклад, клас «*Рахунок*» може перебувати в станах *активний* (звичайний режим доступу), *закритий* (клієнт закрив свій рахунок, але він ще не був виключений з записів банку), *з перевищеним кредитним лімітом* (клієнт перевищив кредитний ліміт за своїм рахунком) і *призупинений* (доступ до рахунка був заблокований з яких-небудь причин).

Виділення подій

Отримавши попередній список станів, треба зайнятися пошуком подій, що викликають переходи між цими станами. Для цього потрібно продумати які зовнішні впливи викликають зміни станів. У багатьох ситуаціях подію можна розглядати як завершення поточної діяльності. Наприклад, якщо стаття перебуває в стані *Рецензування*, перехід з цього стану здійснюється після завершення роботи рецензента. Рішення може бути позитивним (перехід у стан *прийнята*) чи негативним

(перехід устан *відкинута*). Завершення діяльності може викликати альтернативні переходи, які можуть додаватися в процесі вдосконалення моделі. Наприклад, стаття може перейти в стан *Прийнята* зобов'язковими змінами.

Інші події можна побачити, подумавши про те, яким чином об'єкт може потрапити в певний стан.

Всередині стану можуть відбуватися і події, що не викликають переходів. Для моделі станів предметної області важливі тільки ті події, що викликають переходи. Інформацію, що міститься в події, слід подавати у формі списку його параметрів.

Наприклад перерахуємо найважливіші події для класу «*Рахунок*»: *закриття рахунка, перевищення кредитного ліміту, повторне неправильне введення PIN-коду, можлива підробка* та *адміністративні дії*.

Побудова діаграм станів

Наступний крок – розподіл подій за станами, до яких вони належать, додавання переходів, що показують зміни станів, викликані здійсненням події в той час, коли об'єкт знаходиться в певному стані. Якщо подія завершує стан, з цього стану зазвичай буває тільки один перехід в інший стан. Якщо подія ініціює цільовий стан, то потрібно розглянути, в яких станах ця подія може відбуватися, і додати на діаграму переходи з цих станів до цільового стану. Якщо подія має різну дію на різні стани, потрібно додати переходи для кожного з цих станів.

Якщо ж об'єкти класів виконують діяльність при переходах, то треба додати цю діяльність на діаграму.

Наприклад, для класу «*Рахунок*» була створена діаграма станів, наведена на рисунку 3.4.

Перевірка діаграм станів

Останній крок – перевірка кожної моделі станів. Чи всі стани досяжні? Особливу увагу потрібно приділити маршрутам по

діаграмі. Якщо діаграма описує клас з поведінкою, що розвивається, чи є на діаграмі маршрут, що з'єднує початковий стан з кінцевим? Чи присутні на ній очікувані відхилення від основної послідовності? Якщо діаграма описує клас з циклічною поведінкою, чи є на ній основна петля циклу? Чи є тупикові стани, що завершують цикл?

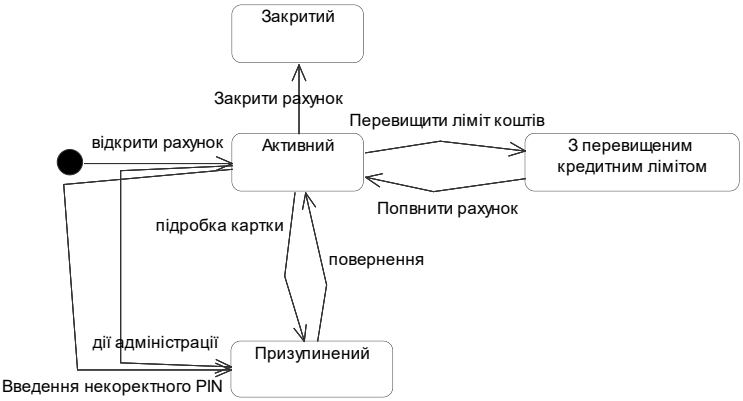


Рисунок 3.4 – Модель станів класу «Рахунок»

Іноді відсутність маршрутів вказує на відсутність потрібних станів. Готова модель станів повинна точно описувати життєвий цикл класу.

3.3 Практичні завдання

Завдання 1. Розгляньте концепцію створення програмного забезпечення для інтернет-сайту лікарні. Дайте відповіді на такі запитання:

Для кого призначений цей додаток? Хто входить до кола зацікавлених осіб? Дайте оцінку потенційних покупців в Україні та за кордоном.

Укажіть три властивості, які повинна мати система, та три властивості, яких не повинно бути.

Вкажіть дві системи, з якими буде працювати ваша система.

Вкажіть три найбільш важливі ризики.

Завдання 2. Підготуйте реферат з методів, що використовуються при підготовці інтерв'ю та анкетування.

Завдання 3. Підготуйте порівняльну характеристику *RAD* та *JAD* технологій.

Завдання 4. Для проєкту системи «Чат-бот аптеки» сформулюйте типові види ризиків.

Завдання 5. Виконайте завдання 1 для системи клінічних досліджень.

Завдання 6. Виконайте завдання 1 для системи моніторингу стану здоров`я пацієнтів.

Завдання 7. Для системи «*Інтернет-магазин ліків»* сформулюйте 20 функціональних вимог, 10 вимог до якості продукту, 10 вимог до інтерфейсу, 5 вимог до апаратного забезпечення.

Завдання 8. Для системи «*Реєстратура лікарні*» сформулюйте 10 функціональних вимог з боку реєстратора, 10 функціональних вимог з точки зору хворого, 10 нефункціональних вимог.

Завдання 9. Розробіть прототипи інтерфейсу для системи «*Реєстратура лікарні*».

Завдання 10. Складіть анкету для виявлення вимог для системи «*Інтернет магазин ліків*».

Завдання 11. Підготуйте огляд методів що використовуються при керування вимогами.

Завдання 12. Розробіть документ Vision для системи «Інтернет-магазин *ліків*».

3.5 Контрольні запитання

1. Що містить аналітична модель?
2. Що відображає модель предметної області?
3. Які підходи використовуються для виявлення класів?
4. Який перелік для класів запропонував Барамі?

5. Чим відрізняється підхід на основі іменних груп від підходу на основі використання прецедентів?

6. Які методи містить підхід CRC?

7. За якими принципами слід знищувати зайві асоціації?

8. Що містить словник даних?

9. Назвіть найбільш стандартні асоціації.

10. За якими правилами слід додавати ти виключати атрибути?

11. За якими двома способами може бути визначено спадкування?

12. Для чого використовують перевірку маршрутів?

13. Як будується модель станів предметної області?

3.6 Література до розділу

1. Mishra J. Software Engineering. [Текст] / J. Mishra, A. Mohanty. – Dorling Kindersley (India), 2012. – 373 p.

2. 20. Schmidt, D.C. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects / D. C. Schmidt,M. Stal,Hans. – USA: John Wilye & Sons, 2013. – 450 p.

3. Griffiths, D. Organizational Learning and Knowledge: Concepts, Methodologies, Tools and Applications – Concepts, Methodologies, Tools and Applications, Volumes 1-4 / D. Griffiths, S. Koukpaki. – USA.: Management Association, Information Resources, 2011. – 3164 p.

4. Буч Г. Язык UML. Руководство пользователя [Текст] /Буч Г. – М.: ДМК Пресс; СПб.: Питер, 2004. – 432 с.

5. Эванс Э. Предметно-ориентированное проєктирование (DDD): структуризация сложных программных систем. [Текст] / Эванс Э. – М.:Вильямс, 2010. – 448 с.

6. Шитікова О.В. Формалізація процесу випробувань газотурбінних установок наземного використання / Шитікова О.В., Табунщик Г.В. // Тиждень науки : тези доповідей науково-практичної конференції, м. Запоріжжя, 13–19 квітня 2012 р./

редкол.: Ю. М. Внуков(відпов. ред.) та ін. – Запоріжжя: ЗНТУ, 2012. -С. .332 – 333.

7. Шитикова Е.В. Информационная модель процесса испытаний газотурбинных установок наземного применения [Текст] / Е.В. Шитикова, Г.В. Табунщик // Радіоелектроніка, інформатика, управління. – 2013. – № 1(28). – С. 101-108. http://ric.zntu.edu.ua/issue/view/777

4 МОДЕЛЮВАННЯ ПОВЕДІНКИ СИСТЕМИ

Поведінка системи – те як вона виглядає для зовнішнього користувача – зображується у вигляді ***прецедентів***. Моделі прецедентів можна розробляти на різних рівнях абстракції. Їх можна застосувати до системи в цілому для того, щоб специфікувати основні функціональні блоки програмного додатка, що розробляється. Їх також можна використати для фіксації поводження пакетів *UML*, частин пакетів або навіть *класу* усередині пакету.

На етапі *аналізу* прецеденти вбирають у себе системні вимоги, концентруючись на тому, *що* робить або повинна робити система. На етапі *проєктування* подання проєктних рішень у вигляді прецедентів можна використати для специфікації поводження системи в тому вигляді, як воно повинне бути реалізоване.

Поводження системи, закріплене за допомогою прецедентів, вимагає здійснити відповідні обчислення й забезпечити взаємодію об'єктів для виконання цих прецедентів. *Обчислення* можна змоделювати за допомогою діаграм видів діяльності. *Взаємодію* об'єктів можна задати за допомогою діаграм послідовностей або діаграм кооперації.

Моделювання поведінки дозволяє подивитися на систему з *погляду її функціонування*. Тут основне завдання полягає в тому, щоб визначити прецеденти для області додатків і встановити, які класи беруть участь у виконанні цих прецедентів. При цьому необхідно ідентифікувати операції класів і повідомлення, передані між об'єктами. Хоча взаємодія об'єктів ініціює зміни стану об'єктів, моделювання поведінки дає функціональний погляд на миттєвий стан системи.

Моделі прецедентів повинні розроблятися ітеративно та паралельно з моделями класів. У ході створення моделей поведінки з'являються ще два рівні класів:

– класи, що обслуговують події, які ініціюються користувачами, та являють собою бізнес-процеси (*керуючі класи*);

– класи, що являють собою *GUI*-інтерфейси (*прикордонні класи*).

4.1 Моделювання взаємодії

Модель взаємодії для додатка будується в декілька етапів:

– виконати моделювання прецедентів;

– побудувати діаграми діяльності для складних прецедентів;

– виконати моделювання взаємодії.

Моделювання прецедентів

Моделювання прецедентів тісно пов'язане з установленням вимог. Вимоги, викладені в текстовому вигляді в документі опису вимог, необхідно довести до прецедентів, зафіксованих у документі специфікації вимог. Якщо подальший процес розроблення керується прецедентами, то процес називається проблемно-орієнтованим.

Подібно до моделювання класів моделювання прецедентів істотно ітеративний та нарощуваний процес. Первісну діаграму прецедентів можна визначити на основі вимог верхнього рівня. Це може бути модель бізнес-прецедентів. Для подальшого уточнення прецедентів варто керуватися більш деталізованими вимогами. Якщо протягом ЖЦ розробки вимоги користувачів піддаються змінам, ці зміни варто спершу занести до документа опису вимог, а вже потім – до моделі прецедентів. Потім зміни в прецедентах доводять до інших моделей.

Ідентифікація дійових осіб. По-перше, необхідно ідентифікувати *зовнішні об'єкти*, що безпосередньо взаємодіють із системою. Це і будуть дійові особи. До їх числа належать люди, зовнішні пристрої та інші програмні системи. Найважливішою властивістю дійових осіб є те, що вони не контролюються програмно, а їх поведінка повинна вважатися непередбачуваною. Попри те, що може існувати якась очікувана послідовність дій

дійової особи, система повинна бути досить стійкою, щоб витримувати порушення цієї послідовності.

У процесі пошуку дійових осіб цікавлять не індивідуальні сутності, а з архетипічною поведінкою. Кожна дійова особа – це ідеалізований користувач, який використовує будь-яку підмножину функціональності системи. Необхідно вивчити кожний зовнішній об'єкт і з'ясувати, чи не характеризується він істотно різними видами поведінки. Дійова особа – це один варіант поведінки відносно до системи, а один зовнішній об'єкт може відповідати кільком дійовим особам. З іншого боку, різні типи зовнішніх об'єктів можуть описуватися однією дійовою особою.

Наприклад, для системи обслуговування клієнтів банку одна людина може одночасно бути і касиром, і клієнтом одного і того ж банку. Це цікавий збіг, який, проте, частіше за все не є важливим. Відносно банку людина завжди виступає або в одній, або в іншій ролі. Для додатка для банкомата дійовими особами будуть клієнт, банк і консорціум.

Ідентифікація прецедентів. При виявленні прецедентів аналітик повинен переконатися в тому, що він твердо дотримується сутності концепції прецедентів. Прецеденти являють собою такі компоненти загальної моделі системи:

– завершений фрагмент функціональних можливостей (включаючи основний потік логіки керування, його будь-які варіації (підпотоки) та виняткові умови (альтернативні потоки));

– фрагмент із зовні спостережуваних функцій (відмінних від внутрішніх функцій);

– ортогональний фрагмент функціональних можливостей (прецеденти можуть при виконанні спільно використати об'єкти, але виконання кожного прецеденту незалежно від інших прецедентів);

– фрагмент функціональних можливостей, що ініціюється суб'єктом (будучи ініційований, прецедент може взаємодіяти з іншими суб'єктами). При цьому можливо, що суб'єкт виявиться

тільки на приймальному кінці прецеденту (може бути опосередковано), ініційованого іншим суб'єктом;

– фрагмент функціональних можливостей, що надає суб'єктові відчутного корисного результату (і цей корисний результат досягається в межах одного прецеденту).

Виявлення прецедентів базується на аналізі наступних джерел інформації:

– вимоги, що визначені в документі опису вимог;

– суб'єктів і їхніх цілей стосовно до системи.

У ході аналізу прецеденти звертаються до особистісних потреб суб'єктів. Будь-яка поведінка системи має належати певному прецеденту. Іноді можуть виникнути проблеми з віднесенням якої-небудь поведінки, близької до межі, до того чи іншого прецеденту. При розбитті завжди слід пам'ятати про те, що завжди існують прикордонні варіанти. У цьому випадку рішення можна прийняти довільно.

Кожний прецедент має відображати якийсь сервіс (послугу), що надається системою, тобто щось цінне для діючої особи. Всі прецеденти слід розглядати на одному рівні деталізації. Наприклад, якщо один з варіантів використання для банку називається *«Запросити позику»*, то інший варіант не повинен називатися *«Зняти готівку з депозитного рахунка через банкомат»*. Друга назва занадто деталізована порівняно з першим. Краще назвати цей варіант просто «*Зняти гроші»*.

Отже, для системи роботи з банкоматом можемо виділити наступні прецеденти:

– *«Ініціалізація сеансу»* – банкомат визначає особу користувача і пропонує йому список дій;

– *«Запитування рахунка»* – система надає загальні відомості про рахунок, такі як поточний баланс, дату останньої операції, дату відправлення останнього звіту поштою;

– *«Оброблення операції»* – система банкомата виконує дії, що впливають на баланс рахунка, такі як внесок, зняття та

переведення коштів. Банкомат гарантує, що всі завершення операцій в кінцевому підсумку записуються в базу даних банку;

– *«Передання даних»* – банкомат використовує можливості консорціуму для взаємодії з комп'ютерами відповідного банку.

Ідентифікація початкових і кінцевих подій. Прецеденти розбивають функціональність системи на дискретні частини та показують діючі особи, які використовують кожну з цих частин. Проте поведінка системи демонструється в них недостатньо чітко. Для розуміння поведінки необхідно знати послідовності виконання, відповідні кожному з прецедентів. Починати їх аналіз слід з пошуку подій, що ініціюють кожний з прецедентів. Треба визначити, яка особа ініціює варіант використання, і подію, яку вона для цього передає системі. У багатьох випадках початковою подією є запит певної послуги, що надається системою. В інших випадках початковою подією є подія (пригода), що запускає ланцюжок дій. Дайте цій події осмислену назву, але поки не намагайтеся визначити повний список її параметрів.

Крім того, слід визначити кінцеву подію або групу подій, а також загальні рамки подій, що повинні бути включені в кожний з прецедентів. Наприклад, прецедент «Запит позики» може тривати до тих пір, поки прохання не буде подано, поки позика не буде видана чи відхилена або поки позика не буде погашена та закрита. Будь-який з цих варіантів цілком прийнятний. Розробник моделі повинен визначити межі прецедента, встановивши його кінцеву подію.

Підготовка типових сценаріїв. Для кожного прецеденту потрібно підготувати один або кілька типових сценаріїв, щоб відчути очікувану поведінку системи. Ці сценарії описують основні взаємодії, формати зовнішніх відображуваних даних, а також обмін інформацією. Сценарієм називається послідовність подій на множині взаємодіючих об'єктів. Аналіз варто здійснювати в термінах прикладів взаємодії, а не розписувати найбільш загальні варіанти.

Для більшості завдань логічна коректність залежить від взаємної послідовності взаємодій, а не від конкретних часових проміжків між ними.

Іноді опис завдання повністю задає послідовність взаємодії, але в більшості випадків її доведеться вигадувати (або, принаймні, конкретизувати). Наприклад, у постановці задачі про банкомат ідеться про необхідність отримання даних про транзакції від користувача, але не вказується, які конкретно параметри потрібно в нього запитати і в якій послідовності.

Для більшості додатків порядок введення вихідних даних не має особливої важливості та може бути відкладений до етапу проєктування.

Підготуйте сценарії для типових ситуацій – взаємодій без незвичайних вхідних параметрів і помилкових ситуацій. Подією є обмін інформацією між об'єктом системи та зовнішнім агентом. Параметром події є передана інформація. Наприклад, параметром події «введений пароль» є сам введений пароль. Події без параметрів теж мають значення і досить поширені. Саме здійснення події є інформацією. Для кожної події необхідно вказати особу, що викликала її (систему, користувача або іншого зовнішнього агента) і параметри події.

Наприклад, для прецеденту «*Ініціалізація сеансу роботи з банкоматом*» можна описати такий сценарій:

Банкомат запрошує користувача вставити кредитну картку.

Користувач вставляє кредитну картку.

Банкомат приймає картку і зчитує її серійний номер.

Банкомат запитує пароль. Користувач вводить «1234».

Банкомат перевіряє пароль, зв'язуючись із консорціумом і банком.

Банкомат виводить меню дій над рахунками та команд.

Для прецеденту «Оброблення операції» основний сценарій буде таким:

Банкомат виводить меню рахунків та команд.

Користувач обирає зняття грошей з рахунка.

Банкомат запитує суму, що знімається.

Користувач вводить суму.

Банкомат перевіряє суму на перевищення ліміту видачі готівки.

Банкомат зв'язується з консорціумом та банком для перевірки, чи достатньо коштів на рахунку.

Банкомат видає готівку та просить користувача її забрати.

Користувач бере готівку.

Банкомат виводить меню дій над рахунками і команд.

Нетипові сценарії та виняткові ситуації. Після розробки типових сценаріїв необхідно розглянути особливі ситуації, такі як відсутність введених значень, введення мінімального та максимального значень, введення однакових значень. Потім потрібно розглянути помилкові ситуації, включаючи введення неправильних значень і відсутність відгуку. Для багатьох інтерактивних додатків оброблення помилок є найбільш складною частиною процесу розроблення. Необхідно надавати користувачу можливість відмінити операцію або відкотитися до чітко визначеної початкової точки кожного етапу. Нарешті, потрібно розглянути додаткові взаємодії, що можуть перетинатися з базовими (такі як звернення до довідкової системи або запит відомостей про стан).

Наприклад, для прецеденту «*Ініціалізація сеансу роботи з банкоматом*» можна виділити такі нетипові сценарії та виняткові ситуації:

Банкомат не може прочитати картку.

Термін дії картки закінчився.

Тайм-аут при очікуванні банкоматом відповіді клієнта.

Лінії зв'язку не працюють.

Для прецеденту «*Оброблення операцій*» :

Введена сума перевищує ліміт видачі готівки

Введена сума перевищує суму коштів на рахунку.

Додаткові сценарії описують роботу адміністративних частин системи банкомата, наприклад авторизацію нових карток.

Структурування дійових осіб і прецедентів. На даному етапі потрібно впорядкувати прецеденти за допомогою відносин включення *(<<include>>)*, розширення*(<<extend>>)* та узагальнення. Це особливо корисно для великих і складних систем. Як і з моделями класів і станів, структурування краще відкласти до тих пір, поки не будуть сформульовані всі базові прецеденти. Якщо виконати структурування занадто рано, існує небезпека спотворення додатка через фіксування підсвідомих міркувань у структурі прецедентів.

Узагальнення можна застосовувати і до дійових осіб. Наприклад, адміністратора можна вважати оператором, що має додаткові привілеї.

Наприклад, на рис. 4.1 наведена діаграма прецедентів, що узагальнено зображає роботу системи банківського обслуговування через банкомати.

Перевірка щодо моделі класів предметної області. На цьому етапі модель предметної області та модель додатку повинні бути вже добре узгоджені одна з одною. Дійові особи, прецеденти та сценарії засновані на класах і концепціях моделі предметної області. Згадайте, що одним з етапів побудови цієї моделі є перевірка маршрутів. На практиці ця перевірка – перший крок до виділення прецедентів.

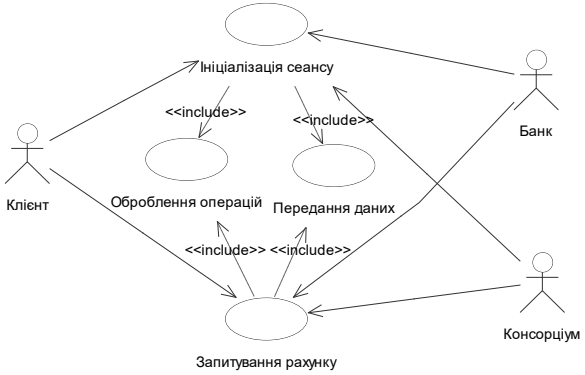


Рисунок 4.1 – Діаграма прецедентів

Зіставте модель додатка та модель предметної області, щоб переконатися в їх узгодженості. Вивчіть сценарії і перевірте, що в моделі предметної області є всі необхідні дані. Переконайтеся, що ця модель містить всі параметри подій.

Моделювання зовнішніх подій

Проаналізуйте всі розроблені сценарії та виділіть всі зовнішні події: введення даних, прийняття рішень, переривання та взаємодії з іншими користувачами та зовнішніми пристроями. Подія може викликати дії цільового об'єкта. Етапи внутрішніх обчислень не є подіями, за винятком розрахунків, у ході яких здійснюється взаємодія із зовнішнім світом. За допомогою сценаріїв ви можете відшукати типові події, але не забудьте і про нетипові події та помилкові ситуації.

Передання інформації об'єкту є подією. Наприклад, «*введений пароль*» – це повідомлення, передане від зовнішнього агента «*Користувач*» об'єкту програми «*Банкомат*». Деякі потоки інформації наявні в моделі неявним чином. Багато подій характеризуються певними параметрами.

Згрупуйте під однаковою назвою події, що надають однаковий вплив на потік управління, навіть якщо значення їх параметрів відрізняються. Наприклад, «*введений пароль*» – це подія, параметром якого є значення пароля. Вибір значення пароля не впливає на потік управління, тому події з різними паролями є екземплярами одного і того ж типу подій. Аналогічним чином, «*видача готівки*» також є подією незалежно від суми (параметра). Екземпляри подій, значення яких впливають на потік управління, повинні бути віднесені до різних типів подій.

Треба стежити за ситуаціями, коли розбіжності кількісних значень досить істотні для того, щоб події можна було вважати різними. Наприклад, натискання будь-якої цифрової клавіші на

клавіатурі можна вважати подією, що не залежить від конкретної цифри, тоді як натискання клавіші «введення» можна розглядати окремо, тому що програма буде обробляти його не так, як натискання на цифрові клавіші. Розрізнення подій залежить від додатка.

Підготуйте діаграму послідовності для кожного сценарію. Діаграма послідовності показує учасників взаємодії та послідовність повідомлень, якими вони обмінюються. Кожному учаснику виділяється свій стовпець. Діаграма показує відправника та одержувача кожного повідомлення. Якщо в сценарії бере участь кілька об'єктів одного і того ж класу, їм слід присвоїти різні номери. Вивчивши один стовпчик таблиці, ви можете визначити події, що безпосередньо впливають на конкретний об'єкт. Після цього ви можете згрупувати події, що відправляються і приймаються кожним класом.

Для кожного альтернативного потоку повинна існувати своя діаграма взаємодії.

Приклад з банкоматом. На рис. 4.2 показана діаграма послідовності для головного сценарію «Оброблення операції», на рис. 4.3 – діаграма для альтернативного потоку.

Діаграми послідовності описують діалог і взаємодію дійових осіб, але на них не можна відобразити наявні альтернативи й прийняті рішення. Для цього використовують окрему діаграму для основного потоку взаємодії та окремі діаграми для кожної помилкової ситуації і кожної точки прийняття рішення. Діаграми діяльності дозволяють об'єднати всю цю поведінку завдяки документуванню розгалужень і злиттів (поєднань) потоку управління.

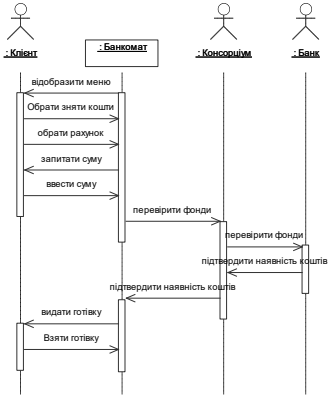


Рисунок 4.2 – Діаграма послідовності для прецеденту «Оброблення операції» для основного сценарію

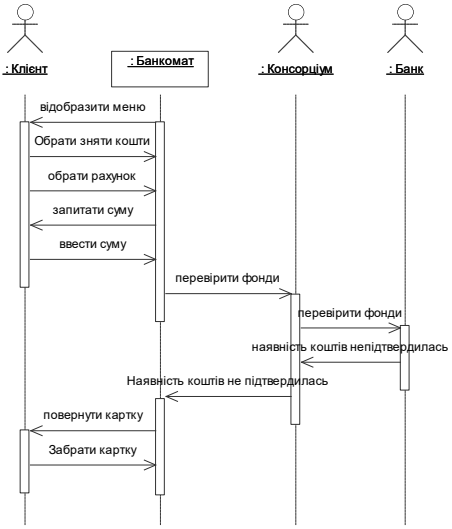


Рисунок 4.3 – Діаграма послідовності для альтернативного потоку прецеденту «Оброблення операції»

Підготовка діаграм діяльності для складних прецедентів

Подібно традиційним потоковим діаграмам і структурним схемам, що одержали поширення в рамках структурних методів для розроблення процедурно-орієнтованих програм, діаграми видів діяльності являють собою потік логіки керування в об'єктно-орієнтованих програмах. Розходження ж полягає в можливості подання за допомогою діаграм видів діяльності керування паралельними потоками поряд з послідовним керуванням.

Моделі видів діяльності широко використаються в проєктуванні.

Однак вони можуть бути особливо корисні для визначення потоків видів діяльності в процесі виконання прецедентів. Оскільки моделі видів діяльності не відображають об'єктів, що здійснюють діяльність, граф видів діяльності можна побудувати навіть у тому випадку, коли модель класів відсутня або розробляється. Зрештою кожний вид діяльності визначається однією або декількома операціями в одному або декількох класах, що кооперуються. Детальне опрацювання такої кооперації може бути здійснене з використанням діаграм кооперації.

Кожний прецедент можна моделювати за допомогою одного або декількох графів видів діяльності. Подія, джерелом якої служить суб'єкт – ініціатор прецеденту, ця та ж сама подія, що запускає виконання графа видів діяльності. Процес виконання послідовно переходить від одного стану виду діяльності до іншого. Стан виду діяльності вважається завершеним, коли завершується його обчислення. Зовнішні переривання, що ініціюються подіями, які можуть викликати завершення стану виду діяльності, допускаються тільки у виняткових випадках. Якщо очікується, що подібні події можуть відбуватися часто, то треба замість цього скористатися діаграмою станів.

Види діяльності найкраще виявляти на основі аналізу пропозицій неформальної специфікації прецедентів. Кожна фраза,

що містить дієслово, може розглядатися як потенційний вид діяльності. Опис альтернативних потоків уводить у граф видів діяльності розгалуження й поділ потоків. Вони приводять до виняткових станів діяльності. Можливі також паралельні потоки керування.

Після виявлення станів видів діяльності специфікація видів діяльності виглядає як досить простий процес з'єднання цих станів лініями переходів. Паралельні потоки ініціюються (розділяються) і зливаються, альтернативні потоки розгалужуються та поєднуються, що відображається у вигляді ромбів розгалуження.

Зовнішні події на графі видів діяльності звичайно відсутні. Однак існує графічний метод включення зовнішніх подій у граф. Аналогічно, існують графічні позначення для станів потоків об'єктів для подання об'єктів, які є вхідними або вихідними для годиться діяльності.

На рисунку 4.4 наведена діаграма діяльності для перевірки картки.

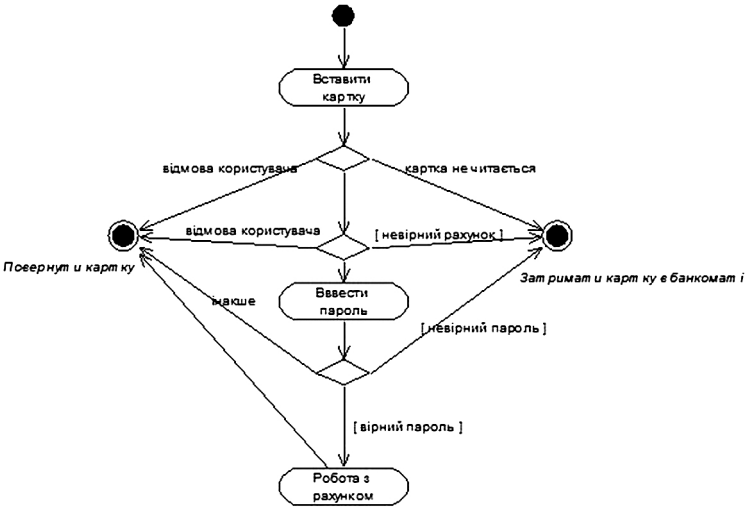


Рисунок 4.4 – Діаграма діяльності для перевірки картки

4.2 Практичні завдання

Завдання 1. Сформулюйте властивості та визначте прецеденти для наступних систем:

1.1 Система «*Електронна медична бібліотека*». Для системи повинні виконуватись такі вимоги: це незалежна бібліотека, бібліотека працює лиш з електронними журналами, користувачі бібліотеки – модератори та читачі. Читачі повинні сплатити реєстрацію за рік – за використання чи певного журналу чи всієї бібліотеки. Користувач може отримати електронну копію або проглядати електрону версію online статті або журналу якщо в нього є передплата. Або придбати додатково якщо раніше передплати не було. Модератор оновлює електронні версії журналів, обробляє повідомлення від читачів, виконує розсилку, коли додається новий журнал, перевіряє інформацію від читачів про реєстрацію та платежі.

1.2 Система «*Склад ліків*». Для системи повинні виконуватись такі вимоги: товар, що зберігається має назву, розрізняється за видом, класом та типом. У кожного товару є унікальний код, вартість за яку був придбаний, остання дата оновлення, кількість одиниць, місце зберігання. Товар прибуває до складу та відбуває зі складу. Кожний вид товару може обслуговувати тільки визначений комірник, у якого є унікальний код у системі. Менеджер може переглянути кількість одиниць товару, відповідального комірника, замовити товар у постачальника якщо товар відсутній, відмовити у постачанні товару, якщо він відсутній. При отриманні товару оформлюється відповідна накладна, що містить кількість одиниць товару, постачальник, вартість одиниці, комірник що отримав, експедитор, що привіз. Виконується оплата товару через банківський переказ. Якщо фактична кількість та кількість у накладній відрізняється, то робиться додатковий заказ. Інформація про товар оновлюється. Накладні зберігаються у загальному реєстрі. Відправлення товару зі складу відбувається відповідно до бланку замовлення. До клієнта можуть бути відправлені декілька видів товарів, що відображається у замовленні. Якщо на складі відсутні необхідна

кількість товару то оформлюється додаткове замовлення. Відправлення виконується тільки після попередньої оплати товару через банківський переказ. Інформація про товар оновлюється. Замовлення зберігаються у загальному реєстрі.

Завдання 2. Для вищезгаданих систем створіть прецеденти та відповідні документи (*Use Case Document*) з використанням *Rational RequisitePro*, намалюйте діаграми прецедентів.

Завдання 3. Для кожного прецедента з попереднього завдання розробіть альтернативні потоки для обробки помилок.

Завдання 4. Розгляньте як зміниться діаграма класів та прецедентів якщо для системи *«Електронна медична бібліотека»* додати такі вимоги:

– система може працювати з електронними книжками;

– книжки можна читати тільки в електронному вигляді з сайту бібліотеки.

Завдання 5. Створіть діаграми взаємодії для кожного основного сценарію та альтернативних потоків.

Завдання 6. Створіть діаграми діяльності для вишевизначених прецедентів.

4.3 Контрольні запитання

1. Яке призначення прецедентів?
2. Які класи з’являються як результат розробки моделі взаємодії?
3. Які етапи потрібно виконати для будування моделі взаємодії?
4. Які компоненти загальної моделі системи являють собою прецеденти?
5. Що відображає прецедент?
6. Які типи сценаріїв використовуються при описі прецедентів?
7. Чи можливо на діаграмах взаємодії одночасно зобразити основний та альтернативний сценарій?
8. Як виконується моделювання зовнішніх подій?

8. Коли рекомендується використовувати діаграми схем діяльності?

4.4 Література до розділу

1. Шитикова Е. В. Модельно-ориентированные методы автоматизации процесса испытаний сложных технических систем / Шитикова Е. В. Табунщик Г. В. // Электротехнические и компьютерные системы № 22 (98), 2016, - С. 338 – 346 http://dx.doi.org/10.15276/eltecs.22.98.2016.61

2. Tabunshchyk G. Flexible Technologies for Smart Campus/ D. Van Merode, G. Tabunshchyk, K. Patrakhalko, Y. Goncharov // Proceedings of XIII International Conference on Remote Engineering and Virtual Instrumentation (REV2016) (24-26 February, 2016, Madrid, Spain) UNED: pp. 58-62.

3. А.С. № 66615 Система керування контентом для віддалених експериментів з дослідження надійності вбудованих систем /Табунщик Г.В., Охмак В.О., опубл. 13.07.2016

4. Tabunshchyk G. Multipurpose Educational System based on Raspberry Pi/ G. Tabunshchyk, D. Van Merode , O.Petrova, V. Okhmak // Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering, Nitra, Slovakia, 12-15 September, 2016 – pp. 202-206

5. Tabunshchyk G. SMART-CAMPUS INFRASTRUCTURE DEVELOPMENT BASED ON BLE 4.0/ G. Tabunshchyk, PhD., D. Van Merode, Y. Goncharov, K. Patrakhalko // Journal Electrotechnic and Computer Systems No. 18 (94), 2015 17 – 20

6. Arras, P. Iterative Pattern for the Embedding of Remote Laboratories in the Educational Process / P. Arras, K. Henke , G.Tabunshchyk, D. V.Merode // 12th International Conference on Remote Engineering and Virtual Instrumentation (REV 2015) 25-28 February 2015, Bangkok, Thailand, PP. 52-55 (10.1109/REV.2015.7087262)

5 ПРОЄКТУВАННЯ СИСТЕМИ

5.1 Основи проєктування систем

У процесі аналізу ми зосереджуємося на тому, що повинно бути зроблено, і не замислюємося про те, як це слід робити. В процесі проєктування розробники приймають рішення про те, яким чином вони будуть вирішувати завдання, спочатку на високому рівні, а потім в деталях.

Проєктування системи – це перший етап проєктування, в процесі якого вибирається базовий підхід до вирішення завдання. Розробники формулюють загальну структуру та стиль рішення.

Архітектура системи визначає її розбиття на підсистеми. Крім того, архітектура створює контекст, у якому приймаються подальші рішення про детальну будову системи. Створення архітектури вимагає послідовного виконання таких дій:

– оцінити продуктивність системи;

– скласти план повторного використання;

– розбити систему на підсистеми;

– виявити властиву завданню паралельність;

– розподілити підсистеми відповідно до апаратного забезпечення;

– установити пріоритети при компромісах;

– спланувати сховища даних;

– розподілити глобальні ресурси;

– вибрати стратегію управління програмним забезпеченням;

– обробити пограничні умови;

– установити пріоритети при компромісах;

– вибрати стиль архітектури.

Оцінка продуктивності

На початковому етапі планування нової системи необхідно провести наближену оцінку її продуктивності. Мета полягає не в тому, щоб отримати високу точність оцінки, а в тому, щоб визначити, чи реально побудувати потрібну систему. Розрахунок повинен бути простим і ґрунтуватися на здоровому глузді.

Планування повторного використання

Повторне використання часто називається основною перевагою об'єктно-орієнтованої технології, але воно не забезпечується одним лише фактом застосування цієї технології. Повторне використання характеризується двома істотно різними аспектами: можна використовувати існуючі речі, а можна створювати нові з розрахунком на повторне використання. Набагато простіше використовувати існуюче, ніж проєктувати нове, орієнтуючись на невідомі перспективи повторного використання. Звичайно, щоб ми зараз могли повторно використовувати якісь сутності, хтось у минулому повинен був їх створити. Суть у тому, що більшість розробників використовують існуюче, і лише невелика їх частина створює нове. Не думайте, що вам доведеться починати роботу з об'єктно-орієнтованими технологіями зі створення нових сутностей. Для цього потрібен великий досвід.

Повторно використовуваними можуть бути *моделі*, *бібліотеки*, *каркаси* та *шаблони*. Найбільш практичним можна назвати повторне використання моделей. Логіка моделі часто буває застосована до безлічі завдань.

Бібліотека (library) – це сукупність класів, які можуть виявитися корисними в безлічі контекстів. Ця сукупність повинна бути добре влаштована, щоб користувачі могли шукати потрібні їм класи. Щоб побудувати таку бібліотеку, потрібно багато зусиль, і часто буває важко вирішити, де має бути розміщений який-небудь об'єкт. У вирішенні цього завдання може допомогти мережевий пошук, але він не замінить хорошої

організації бібліотеки. Крім того, класи повинні мати точний та докладний опис, щоб користувачі могли самі вирішити, підходять вони їм чи ні. Відзначаються наступні якості, якими повинні володіти «хороші» бібліотеки класів:

– цільність – бібліотека класів повинна охоплювати невелику кількість чітко визначених тем;

– повнота – бібліотека класів повинна повністю описувати поведінку за обраними темами;

– узгодженість – поліморфні операції повинні характеризуватися однаковими іменами та сигнатурами в різних класах;

– ефективність – бібліотека повинна надавати альтернативні реалізації алгоритмів (таких як алгоритми сортування), що дозволяють вибирати між швидкістю й обсягом пам'яті;

– розширюваність – користувач повинен мати можливість визначати підкласи класів бібліотеки;

– універсальність – скрізь, де це можливо, повинні використовуватися параметризовані визначення класів.

Каркас (framework) – це скелет програми, деталізація якого дозволяє створити повноцінний додаток. Деталізація найчастіше зводиться до конкретизації абстрактних класів поведінкою, специфічною для даного додатку. З каркасом може поставлятися бібліотека класів, завдяки наявності якої програміст може конкретизувати абстрактні класи, вибираючи відповідні підкласи з бібліотеки, а не програмуючи їх поведінку з нуля. Каркас складається не тільки з класів, він включає парадигму потоку управління та загальні інваріанти. Зазвичай каркаси створюються для цілої категорії додатків, а бібліотеки класів для них ще більш спеціалізовані і не підходять для вирішення спільних завдань.

Шаблон (pattern) – це апробоване вирішення загального завдання. Різні шаблони призначені для різних стадій циклу розробки програмного

забезпечення. Існують шаблони для аналізу, архітектури, проєктування та реалізації. Краще використовувати існуючі шаблони, ніж заново винаходити рішення з нуля. Шаблони зазвичай супроводжуються рекомендаціями щодо їх використання, а також характеристиками для порівняння.

Використання шаблонів дає безліч переваг. Одне з них полягає в тому, що шаблон розроблявся іншими людьми та успішно застосовувався ними на практиці. Отже, він з більшою ймовірністю виявиться коректним і стійким, ніж власне рішення, що не пройшло тестування. Крім того, працюючи з шаблонами, ви вчите язик, знайомий багатьом іншим розробникам. Величезна кількість книг описує шаблони, тонкощі та нюанси роботи з ними. Можна розглядати шаблони як розширення мови моделювання. Не обов'язково мислити в термінах примітивів, можна оперувати їх комбінаціями. Шаблони – це фрагменти моделі, що виражають знання експертів.

Шаблон відрізняється від каркасу. Перший зазвичай складається з невеликої кількості класів, зв'язаних відносинами. Навпаки, каркас зазвичай містить на порядок більше класів і охоплює цілу підсистему або навіть додаток.

Шаблони поділяються на:

– шаблони розподілення обов'язків (*software design*);

– архітектурні (architectural) шаблони (*systems design*);

– шаблони проєктування (*GoF*);

– ідіоми – idioms (*low level*);

– шаблони аналізу – analysis patterns (*recurring & reusable analysis models*);

– шаблони архітектури корпоративних програмних додатків (*patterns of Enterprise Application Architecture*);

– шаблони організації (*structure of organizations/projects*);

– шаблони процесів (*software process design*).

Розбиття системи на підсистеми

Для всіх додатків, за винятком найпростіших, перший етап проєктування системи зводиться до розподілу цієї системи на частини. Кожна велика частина системи називається підсистемою. Кожна підсистема володіє чимось таким, що відрізняє її від інших підсистем, наприклад єдиною функціональністю, фізичним розміщенням або виконанням на однотипному обладнанні. Наприклад, комп'ютер космічного корабля може складатися з підсистем життєзабезпечення, навігації, управління двигунами та проведення космічних експериментів.

Підсистема (*subsystem*) – це не об'єкт і не функція, а сукупність класів, асоціацій, операцій, подій та обмежень, зв'язаних між собою і які мають добре визначений та не надто обширний інтерфейс з іншими системами. Підсистема зазвичай визначається сервісами, які вона надає. *Сервіс* (*service*) – це група споріднених функцій, що мають загальне призначення, наприклад оброблення вводу-виводу, побудова малюнків і виконання арифметичних операцій. Підсистема виражає певний цілісний погляд на одну частину завдання. Наприклад, файлова система є підсистемою операційної системи. Вона складається з набору зв'язаних абстракцій, які практично не залежать від абстракцій в інших підсистемах (таких як управління пам'яттю і процесами).

Кожна підсистема зв'язана з іншими частинами системи добре визначеним інтерфейсом. *Інтерфейс* визначає форму всіх взаємодій та потоки інформації через границі підсистеми, але він не обмежує внутрішню реалізацію підсистеми. Завдяки цьому кожна підсистема може розроблятися незалежно від інших.

Підсистеми слід визначати таким чином, щоб взаємодії в системі в цілому носили переважно внутрішній характер, тобто здійснювалися всередині підсистем. Це скоротить залежності між різними підсистемами. Кількість підсистем не повинна бути великою: в ролі розумного обмеження можна взяти число 20.

Кожна підсистема, у свою чергу, може бути розбита на підсистеми більш низького рівня.

Відносини між двома підсистемами можуть бути організовані за принципом клієнт-сервер або ж ці підсистеми можуть бути одноранговими. У першому випадку (відношення клієнт-сервер) клієнт викликає сервер, який виконує деякий сервіс і повертає результат. Клієнт повинен знати інтерфейс сервера, але серверу не обов'язково знати інтерфейси клієнта, оскільки ініціатором взаємодії завжди є клієнт.

В однорангових відносинах кожна підсистема може звернутися до будь-якої іншої. Взаємодія підсистем не обов'язково супроводжується негайною відповіддю. Однорангові взаємодії складніші, оскільки підсистеми повинні знати інтерфейси всіх інших підсистем. У них можуть виникати цикли, важкі для розуміння і проєктування. Намагайтеся розбивати систему на клієнтські та серверні частини завжди, коли це можливо, тому що односторонню взаємодію набагато простіше проєктувати, розуміти та змінювати, ніж двосторонню.

Розбиття системи на підсистеми може бути організовано у вигляді послідовності *горизонтальних шарів* або *вертикальних відділів*.

Багатошарова система (*layered system*) – це впорядкована безліч віртуальних шарів (ярусів), кожен з яких будується в термінах шарів, що знаходяться нижче нього, і утворює базис для реалізації вищих шарів. Об'єкти кожного шару можуть бути незалежними, але часто між об'єктами різних шарів спостерігається деяка відповідність. Знання в даному випадку є односторонніми: підсистема знає про нижні шари, але не про верхні. Між верхніми і нижніми шарами є відношення клієнт-сервер.

Наприклад, в інтерактивній графічній системі вікна будуються за допомогою операцій з екраном, що складаються з операцій з окремими пікселами, які зводяться до операцій

введення-виведення з пристроями. Кожний шар може мати свій набір класів і операцій. Кожний шар реалізується в термінах класів та операцій нижчих шарів.

Багатошарова архітектура буває двох видів: відкрита і закрита. У закритій архітектурі кожний шар будується тільки в термінах того рівня, який знаходиться безпосередньо під ним. Це скорочує залежності між шарами та полегшує внесення змін, оскільки інтерфейс шару впливає тільки на наступний за ним шар. У відкритій архітектурі шар може використовувати функції будь-якого нижчого рівня. Це скорочує потребу в перевизначенні операцій на кожному рівні, завдяки чому код може вийти більш ефективним і компактним. Однак відкрита архітектура не сприяє прихованню інформації. Зміни в підсистемі можуть вплинути на будь-яку вищу за підпорядкуванням підсистему, тому така архітектура виходить менш стійкою, ніж закрита. Обидва типи корисні. Проєктувальнику доводиться зважувати переваги і недоліки кожної з них.

Зазвичай у постановці завдання визначаються тільки верхній і нижній рівні. Верхній рівень – це потребуєма система, а нижній – це доступні ресурси (обладнання, операційна система, існуючі бібліотеки). Якщо різниця між ними дуже велика (а так буває досить часто), вам доводиться вводити додаткові рівні для скорочення концептуальних розривів між сусідніми рівнями.

Багатошарову систему можна перенести на інше обладнання, переписавши тільки один з її рівнів. Потрібно намагатися вводити принаймні один рівень абстрагування між додатком і будь-якими сервісами, що надаються операційною системою чи обладнанням. Додайте рівень класів інтерфейсу, що надають логічні сервіси, і відобразіть їх на конкретні системно-залежні сервіси.

Розділи (partitions) ділять систему по вертикалі на кілька незалежних або слабко зв’язаних між собою підсистем, кожна з яких надає сервіси одного типу. Наприклад, до складу комп’ютерної операційної системи входять

файлова система, диспетчер процесів, система управління віртуальною пам'яттю та система керування пристроями. Підсистеми можуть знати про існування інших підсистем, але це знання не є глибоким і не створює серйозних залежностей на рівні проєктування.

Різниця між шарами та розділами полягає в тому, що шари відрізняються один від одного рівнем абстракції. Розділи ж просто ділять систему на частини, що знаходяться на одному і тому ж рівні абстракції. Ще одна відмінність полягає в тому, що шари обов'язково залежать один від одного (зазвичай за типом клієнт-сервер). Розділи утворюють однорангову систему і є відносно незалежними або взаємозалежними.

Ви можете розбити систему на підсистеми, поєднуючи шари та розділи. Рівні можуть розбиватися на розділи, а розділи – на рівні.

Після визначення підсистем вищого рівня необхідно показати інформаційні потоки. Іноді всі підсистеми взаємодіють між собою, але частіше за все структура потоків виявляється більш простою. Наприклад, обчислення часто організуються у вигляді конвеєра (прикладом такої структури є компілятор). Інші системи мають структуру зірки, в якій головна підсистема керує всіма взаємодіями інших підсистем. Скрізь, де це можливо, слід прагнути спрощувати топологію і скорочувати кількість взаємодій між підсистемами.

Виділення паралелізму

В аналітичній моделі, у реальному світі та в апаратному забезпеченні всі об'єкти діють паралельно один з одним. Однак у програмній реалізації системи не всі об'єкти виявляються паралельними, тому що один процесор може і не забезпечувати одночасне виконання багатьох об'єктів. На практиці ви можете реалізувати кілька об'єктів на одному процесорі тільки в тому випадку, якщо ці об'єкти не можуть бути активними одночасно.

Одним із важливих завдань етапу проєктування системи є виділення об'єктів, що повинні бути активними одночасно, а також об'єктів, активність яких є взаємовиключною. Об'єкти другої групи можна об'єднати в один потік управління (завдання).

Основним засобом для пошуку паралелізмів є модель станів. Два об'єкти є невід'ємно паралельними, якщо вони можуть одночасно отримувати інформацію про події, не взаємодіючи один з одним. Якщо події не синхронізовані одна з одною, ви не можете об'єднати ці об'єкти в один потік управління. Наприклад, підсистеми управління двигуном і крилами літака повинні працювати паралельно (хоч і не повністю незалежно одне від одного). Звичайно, найкраще працювати з незалежними підсистемами, тому що їх можна розподілити по різних апаратних пристроях без будь-яких витрат на взаємодію.

Не обов'язково реалізовувати невід'ємно паралельні підсистеми у вигляді різних апаратних пристроїв. Найчастіше об'єкти, що мають бути реалізовані у вигляді окремих апаратних пристроїв, перераховуються в описі завдання.

Всі об'єкти з концептуальної точки зору є паралельними, однак на практиці об'єкти часто залежать один від одного. Вивчаючи діаграми станів окремих об'єктів і взаємодію між ними, що здійснюється за допомогою подій, можна об'єднувати групи об'єктів в один потік управління. ***Потік керування*** (*thread of control*) – це маршрут, що проходить через кілька діаграм станів, на якому лише один об'єкт може бути активним у конкретний момент часу. Потік залишається всередині однієї діаграми станів до тих пір, поки об'єкт не надішле подію іншому об'єкту та не перейде в стан очікування іншої події. Тоді потік управління переходить до одержувача події і залишається у нього до тих пір, поки не повернеться до вихідного об'єкту разом з новою подією. Потік може розділитися в тому випадку, якщо об'єкт відправить подію та продовжить своє виконання.

Тільки один об’єкт може бути активним в кожному потоці управління в конкретний момент часу. Потоки управління в комп’ютерних системах реалізуються у вигляді задач.

Розподіл підсистем

Паралельні підсистеми повинні бути розподілені по апаратних пристроях: універсальним процесорам або спеціалізованим функціональним блокам. Для цього потрібно:

– оцінити потреби кожної підсистеми в обчислювальних ресурсах і обсяг ресурсів, необхідний для задоволення цих потреб;

– вибрати апаратну чи програмну реалізацію підсистеми;

– розподілити програмні підсистеми по процесорах з урахуванням вимог, що пред’являються до продуктивності, так щоб мінімізувати взаємодію між процесорами;

– визначити зв’язність фізичних модулів, що реалізують підсистеми.

Рішення використовувати кілька процесорів чи інших апаратних модулів приймається на підставі відомостей про потреби в апаратних ресурсах, що перевищують можливості одного процесора. Кількість процесорів залежить від обсягу обчислень і швидкості комп’ютера.

Проєктувальник системи повинен оцінити необхідну потужність процесора, визначивши стаціонарну нагрузку (добуток кількості транзакцій у секунду на час оброблення транзакції). Наближене значення часто виявляється неточним. Буває корисно провести кілька експериментів. Оціночне значення потрібно дещо завищити, щоб врахувати перехідні ефекти, зв’язані з випадковими флуктуаціями навантаження та синхронізованими сплесками активності. Надлишкова ємність залежить від прийнятної частоти відмов, що викликані нестачею ресурсів. Ви повинні враховувати не тільки стаціонарну (середню) навантаження, але й пікову.

Об’єктно-орієнтований підхід зручний для аналізу апаратних пристроїв. Кожен пристрій – це об’єкт, що існує паралельно з іншими об’єктами (пристроями або програмами). Ви повинні розподілити підсистеми по апаратних пристроях і програмних комплексах. Існує дві причини, з яких певна підсистема може бути реалізована у вигляді апаратного пристрою.

Перша – вартість. Існуюче обладнання повинно надавати необхідну функціональність. Інша – продуктивність. Система вимагає більш високої продуктивності, ніж може надати універсальний процесор, і при цьому існує більш ефективне обладнання.

Складність проєктування системи зв’язана з тим, що ви повинні виконати всі зовнішні вимоги до обладнання та програмного забезпечення. Об’єктно-орієнтоване проєктування – це не чарівна паличка, що може вирішити всі проблеми такого роду, проте воно допомагає моделювати зовнішні пакети. Потрібно враховувати питання сумісності, вартості і продуктивності. Крім того, потрібно зробити систему гнучкою в розрахунку на зміни в майбутньому (як проєктні зміни, так і щодо вдосконалення готового продукту). Гнучкість вимагає певних витрат, і саме архітектор повинен вирішувати, скільки він готовий на неї витратити.

Проєкт системи повинен описувати розподіл програмних підсистем по процесорах. Існує кілька причин, через які цей розподіл має бути формалізовано:

1. Логістика. Деякі завдання повинні виконуватися в певних місцях, тому що вони зв’язані з керуванням, обладнанням або вимагають паралельного виконання. Наприклад, на робочій станції інженера повинна бути встановлена своя власна операційна система, щоб оператор міг працювати з нею в разі перебоїв у мережі.

2. Обмеження на взаємодію. Час відгуку та потік даних перевищують доступну полосу пропускання між задачею та

апаратним пристроєм. Наприклад, високошвидкісні графічні пристрої вимагають установлення власних контролерів, тому що вони породжують великі обсяги даних.

3. Обмеження на обчислювальні ресурси. Якщо потреби в обчислювальних ресурсах занадто великі, щоб їх міг задовольнити один процесор, можна встановити кілька процесорів. Витрати на взаємодію можна скоротити, призначивши задачі, що активно обмінюються даними, одному процесору. Незалежні підсистеми слід розподіляти по різним процесорам.

Визначивши види та кількість апаратних пристроїв, необхідно описати їх розташування і з'єднання між ними. Це містить наступні види робіт:

1. Вибрати топологію для з'єднання фізичних пристроїв. Фізичним з'єднанням часто відповідають асоціації моделі класів. Відносини клієнт-сервер також реалізуються у вигляді фізичних з'єднань. Деякі з'єднання можуть бути опосередкованими. Вартість найбільш важливих відносин слід мінімізувати.

2. Вибрати топологію дублюючих один одного блоків. Якщо потрібно підвищити продуктивність, включивши в систему кілька однотипних модулів, то необхідно вказати і їх топологію. Модель класів тут не допоможе, тому що дублювання блоків зазвичай застосовується для оптимізації на етапі проєктування. Топологія дублюючих блоків зазвичай має регулярну структуру, наприклад: лінійна послідовність, матриця, дерево, зірка. Потрібно врахувати очікувані шаблони надходження даних і запропонувати паралельні алгоритми їх оброблення.

4. Вибрати форму каналів комунікації та протоколи взаємодії. Точні інтерфейси модулів на етапі проєктування враховувати не потрібно, але загальні механізми взаємодії потрібно вказати. Наприклад, взаємодія може бути синхронною, асинхронною або блокуючою. Потрібно оцінити пропускну здатність і затримку каналів комунікації та вибрати найбільш відповідні типи з'єднань.

Навіть якщо з'єднання є логічними, а не фізичними, їх все одно треба розглянути. Наприклад, блоки можуть являти собою завдання, що виконуються в рамках однієї операційної системи і зв'язані між собою засобами міжпроцесної взаємодії (IPC). У більшості операційних систем виклики IPC виконуються набагато повільніше, ніж виклики підпрограм всередині однієї задачі, тому їх використання в задачах з жорсткими часовими обмеженнями може бути непрактично. У цьому випадку вам доведеться об'єднати жорстко зв'язані задачі в одну і реалізувати з'єднання як виклики підпрограм.

Керування сховищами даних

Існує декілька способів зберігання даних, що можуть використовуватися незалежно чи спільно: структури даних, файли і бази даних. Різні варіанти зберігання володіють різними сполученнями вартості, часу доступу, ємності й надійності. Наприклад, додаток на персональному комп'ютері може працювати зі структурами даних у пам'яті з файлами. Система бухгалтерського обліку може використовувати базу даних для взаємодії підсистем. Файли – це дешевий, простий і надійний засіб збереження даних. Однак операції з файлами є низькорівневими, тому в додаток доводиться включати додатковий код, що забезпечує перехід на необхідний рівень абстракції. В різних операційних системах файли реалізуються по-різному, тому додатки, що будуть переноситись, потребують акуратної ізоляції рівня файлової системи. Реалізації файлів з послідовним доступом стандартизовані найкраще, а ось команди та формати зберігання файлів з довільним доступом сильно залежать від системи. Нижче перераховані типи даних, що найкраще зберігати у файлах:

– дані великого обсягу та низької інформаційної щільності (архівні файли, записи журналів);

– середні за обсягом дані з простою структурою;

– дані, доступ до яких здійснюється послідовно;

– дані, що можна повністю завантажити у пам’ять.

Ще один варіант збереження даних – бази даних, керовані системами управління базами даних (СУБД). СУБД (у тому числі реляційні та об’єктно-орієнтовані) випускаються різними виробниками. Вони можуть кешувати дані, що часто використовуються, в пам’яті для оптимізації вартості та продуктивності оперативної пам’яті та дискового простору. Бази даних полегшують перенесення додатків на інше обладнання та на інші платформи, тому що перенесення коду СУБД здійснюється його постачальником. Одним з недоліків СУБД є їх складний інтерфейс. Багато мов баз даних погано інтегруються з мовами програмування. Нижче перераховані типи даних, що найкраще зберігати в базах даних:

– дані, що вимагають детального оновлення безліччю користувачів;

– дані, з якими повинні працювати кілька додатків;

– дані, що вимагають координованого оновлення за допомогою транзакцій;

– великі обсяги даних, що вимагають ефективної обробки;

– дані, що потребують тривалого зберігання та мають велику цінність для організації;

– дані, що повинні бути захищені від несанкціонованого доступу та від зловмисників.

Об’єктно-орієнтовані бази даних не змогли завоювати популярність на широкому ринку. Тому їх слід розглядати тільки в тому випадку, якщо ви розробляєте особливий додаток, що працює з даними безлічі типів або потребує звернення до низькорівневих примітивів для керування даними. До таких додатків відносяться інженерні, мультимедійні додатки, бази знань та електронні пристрої з вбудованим програмним забезпеченням. Для більшості додатків досить реляційної бази даних. Такі бази домінують на ринку, їх функціональність цілком достатня для більшості додатків. При правильному використанні

вони дозволяють дуже добре реалізувати об'єктно-орієнтовану модель.

Розподіл глобальних ресурсів

Проєктувальник системи повинен вказати глобальні ресурси та визначити механізми управління доступу до них. Глобальні ресурси бувають декількох видів:

– фізичні пристрої: процесори, накопичувачі, а також супутники зв'язку;

– простір: дисковий простір, екран робочої станції, кнопки миші;

– логічні назви: ідентифікатори об'єктів, імена файлів, назви класів;

– доступ до загальних даних: бази даних.

Якщо ресурс являє собою фізичний об'єкт, то він може самостійно контролювати доступ до себе за допомогою протоколу керування доступом. Якщо ж ресурс являє собою логічну сутність, то виникає небезпека конфлікту доступу. Наприклад, один і той же ідентифікатор об'єкта може бути одночасно використаний паралельними завданнями.

Уникнути такого конфлікту можна за допомогою «охоронного» об'єкту, якому буде належати кожен глобальний ресурс і який буде керувати доступом до свого ресурсу. Один охоронний об'єкт може керувати кількома загальними ресурсами. Доступ до будь-якого загального ресурсу можливий тільки через його охоронний об'єкт. Віднесення глобального ресурсу до певного об'єкта – це визнання індивідуальності ресурсу.

Ви можете виконати логічне розбиття ресурсу, розподіливши його частини по різним охороняючим об'єктам, завдяки чому буде реалізований паралельний доступ до частин ресурсу. Наприклад, у паралельному розподіленому середовищі, однією зі стратегій виділення ідентифікаторів об'єктів, є попередній розподіл діапазонів можливих ідентифікаторів між процесорами, що

входять до складу мережі. Кожен процесор виділяє ідентифікатори з наданого йому діапазону, завдяки чому зникає потреба у глобальній синхронізації.

У додатках, що ставлять жорсткі вимоги до тимчасових обмежень, вартість доступу до ресурсу через охоронний об'єкт іноді виявляється занадто висока. Клієнти в таких системах повинні працювати з ресурсом напряму. У цьому випадку блокування може бути встановлено на окремі частини ресурсу. *Блокування* – це логічний об'єкт, зв'язаний з деякою підмножиною ресурсу, що дає власникові блокування право на безпосередній доступ до ресурсу. Охороняючий об'єкт, в цьому випадку теж повинен існувати, він займається видачею та зняттям блокування, однак після однієї операції взаємодії з охороним об'єктом для отримання блокування, клієнт може працювати з ресурсом напряму. Такий підхід небезпечніший, тому що кожен користувач ресурсу сам відповідає за свою поведінку під час роботи з цим ресурсом. Не використовуйте прямий доступ до загальних ресурсів, якщо є інші варіанти.

Вибір стратегії керування програмним забезпеченням

В аналітичній моделі взаємодії уявляються як події, що передаються між об'єктами. Апаратне забезпечення в цьому відношенні близько аналітичній моделі, а от у програмному забезпеченні керування може бути реалізовано кількома способами. Не обов'язково, щоб усі підсистеми використовували одну й ту саму реалізацію, але краще всього вибирати для всієї системи єдиний стиль керування. Існує два основних варіанти керування програмною системою: зовнішнє керування та внутрішнє керування.

Зовнішнє керування (*external control*) – це потік видимих ззовні подій між об'єктами системи. Існує три варіанти керування, здійснюваного через зовнішні події: процедурне послідовне, подієве послідовне і паралельне.

Оптимальний вид керування залежить від доступних ресурсів (надаються мовою програмування і операційною системою), а також від видів взаємодії всередині додатку.

Внутрішнє керування (internal control) – це потік керування всередині процесу. Воно з'являється тільки на етапі реалізації, а тому не можна сказати, що йому від початку притаманні паралельність або послідовність. Проєктувальник може розбити процес на кілька задач для забезпечення логічної ясності або для підвищення продуктивності (у разі наявності декількох процесорів). На відміну від зовнішніх подій внутрішні переходи управління (наприклад, виклики процедур або виклики між задачами) знаходяться під контролем програми та можуть бути структуровані так, як зручно. Широко поширені три типи операцій передання керування: виклики процедур, квазіпаралельні виклики між задачами та паралельні виклики між задачами. *Квазіпаралельні завдання* (співпрограми чи програмні потоки) – це конструкції, розроблені для зручності програміста. Кожний програмний потік має власний стек та адресний простір, але тільки один з них може бути активним у конкретний момент часу.

У послідовній системі з *процедурним керуванням* хід виконання визначається кодом програми. Процедури запитують зовнішні дані та чекають на їх надходження. Коли вхідні дані надходять до системи, виконання програми поновлюється з тієї процедури, що запрошувала ці дані. Значення лічильника команд, вміст стека викликів процедур і значення локальних змінних у цьому випадку визначають стан системи.

Основна перевага процедурного керування полягає в тому, що його легко реалізувати в більшості широко поширених мов. Недолік же в тому, що цей варіант потребує відображення Властивої об'єктам паралельності на послідовний потік управління. Проєктувальник повинен перетворювати події в операції між об'єктами. Зазвичай операція відповідає парі подій:

події виведення, що виконує виведення та запитує введення, і події введення, що доставляє до системи нові значення. Ця парадигма незручна для опису асинхронного введення, тому що програма має явно запитувати вхідні дані. Процедурне управління годиться тільки в тому випадку, якщо модель станів демонструє регулярне чергування подій введення та виведення. Більш гнучкі інтерфейси користувача та системи керування побудувати на основі цієї парадигми досить складно.

Зверніть увагу, що основні об'єктно-орієнтовані мови, такі як *C ++* і *Java,* є процедурно-орієнтованими. Нехай вас не вводить в оману словосполучення «передача повідомлень». *Повідомлення* – це виклик процедури з вбудованим оператором case, що залежать від класу цільового об'єкта. Основний недолік звичайних об'єктно-орієнтованих мов у тому, що вони не підтримують властиву об'єктам паралельність. Існують і паралельні об'єктно-орієнтовані мови, але вони поки поширені недостатньо широко.

У послідовній системі, керованій подіями, контроль здійснює диспетчер або монітор, що надається мовою чи операційною системою або виконаний у вигляді окремої підсистеми. Розробники зв'язують процедури-додатки з подіями, і диспетчер викликає ці процедури, коли відбуваються відповідні їм події. Процедури передають диспетчеру вихідні дані або дозволяють введення вхідних даних, але вони не блокуються в очікуванні їх надходження. Всі процедури повертають управління диспетчеру (а не зберігають його до тих пір, поки не будуть отримані будь-які дані). Тому стан системи не може бути описано лічильником програми і стеком. Процедури повинні використовувати глобальні змінні для збереження інформації про стан або ж локальна інформація про стан повинна підтримуватися диспетчером. Подієве керування реалізувати на стандартних мовах складніше, ніж процедурне, але результат у багатьох випадках вартий витрачених зусиль.

Керовані подіями системи виходять більш гнучкими, ніж процедурні. Такі системи моделюють співробітництво процесів усередині однієї багатопоточної задачі. Неправильно написана процедура може заблокувати всі додатки, тому розробникам доводиться бути уважними. Особливо зручними виходять подієві підсистеми інтерфейсу користувача.

Використовуйте керовану подіями систему для зовнішнього керування замість процедурної скрізь, де це можливо, тому що перетворення подій в програмні конструкції в цій парадигмі виконувати простіше. Такі системи виходять «більш модульними» і краще обробляють аварійні ситуації, ніж процедурні системи.

У *паралельній системі* управління здійснюється кількома незалежними об'єктами паралельно. Кожний з таких об'єктів являє собою окрему задачу. У цій системі події реалізуються як односторонні повідомлення (не такі, як «повідомлення» в об'єктно-орієнтованих мовах) між об'єктами. Задача може очікувати введення даних, але інші задачі при цьому продовжують виконуватися. Операційна система планує виконання задач і надає механізм постановки подій в чергу, тому події не губляться в тому випадку, якщо задача у момент їх надходження зайнята чимось іншим. У разі наявності декількох процесорів різні задачі виконуються дійсно паралельно.

Внутрішнє керування. У процесі проєктування розробник розкриває операції на об'єктах у вигляді послідовностей операцій більш низького рівня на тих же самих чи інших об'єктах. Внутрішні взаємодії об'єктів багато в чому схожі зі зовнішніми взаємодіями, тому що для них можуть використовуватися ті ж самі механізми реалізації. Однак існує й важлива відмінність: зовнішні взаємодії завжди мають на увазі очікування подій, тому що об'єкти не залежать один від одного і не можуть змусити один одного відповідати саме в потрібний момент. Внутрішні операції об'єктів являють собою частину алгоритму реалізації, тому їх відгук може бути передбаченим. Отже, внутрішні взаємодії часто можна

розглядати у вигляді викликів процедур: активний об’єкт надсилає запит і чекає на нього відповіді. Існують алгоритми паралельної обробки, проте дуже багато обчислювальних задач добре вирішуються послідовними алгоритмами та можуть бути укладені в єдиний потік керування.

Існують і інші парадигми. Наприклад, бувають системи, засновані на правилах, системи логічного програмування й інші форми програм, що не мають процедур. Вони висловлюють інший стиль керування: явне керування замінюється декларацією тверджень і неявними правилами обчислень, що можуть бути недетермінованими або складними. На сьогодні такі мови використовуються у вузьких областях, таких як штучний інтелект та програмування баз знань, але з часом вони можуть почати застосовуватися ширше.

Облік граничних умов

Проєкт системи здебільшого визначається її поведінкою в стаціонарному режимі, проте обов’язково потрібно розглянути пограничні умови та врахувати такі моменти, як ініціалізацію, завершення та відмову системи.

Система повинна перейти зі статичного початкового стану в прийнятний стаціонарний стан. Вона повинна виконати ініціалізацію констант, параметрів, глобальних змін, задач, охороняючі об’єкти та, можливо, навіть саму ієрархію класів. У процесі ініціалізації лише невелика частина функціональності системи зазвичай буває доступною. Особливо складна ініціалізація системи з паралельними задачами, тому що на цьому етапі незалежні об’єкти не повинні піти далеко вперед або відстати від усіх інших.

Завершення зазвичай виявляється простішим ініціалізації, тому що більшість об’єктів можна просто знищити. Задача повинна звільнити всі зарезервовані нею зовнішні ресурси. У

паралельній системі задача повинна повідомляти про своє завершення всі інші задачі.

Відмова – це незапланований завершення системи. Відмова може бути викликана помилками користувача, проблемами системних ресурсів або зовнішньою поломкою. Проєктувальник системи повинний враховувати можливість її відмови. Відмова може бути викликана й помилками всередині системи та часто виявляється як «неможливе» протиріччя. В ідеальній системі таких помилок бути не може, проте хороший проєктувальник повинен передбачити можливість коректного завершення у разі фатальних помилок. Система повинна залишити середовище виконання якомога більш «чистим» і зберегти якомога більше інформації про причини відмови перед тим, як остаточно завершити роботу.

Встановлення пріоритетів

Проєктувальник системи повинен установити пріоритети, якими потрібно буде керуватися на наступних етапах розробки. Пріоритети вирішують конфлікти між бажаними, але несумісними цілями. Наприклад, частіше за все роботу системи можна прискорити, додавши до неї пам'яті, але при цьому вона буде споживати більше енергії та коштувати дорожче. Компроміси зв'язані не тільки з самою програмою, але і з процесом її розроблення. Іноді буває потрібно пожертвувати повнотою функціональності для того, щоб у потрібний момент випустити продукт на ринок. Іноді пріоритети вказуються при постановці задачі, але частіше відповідальність за суміщення несумісного лягає на проєктувальника.

Проєктувальник повинен визначити відносну важливість різних критеріїв, на підставі чого можна буде надалі йти на компроміси. Проєктувальник не зобов'язаний приймати всі рішення відразу, він просто встановлює правила, за якими ці рішення будуть прийматися.

Рішення проєктувальника впливають на систему в цілому. Успіх чи провал кінцевого продукту може залежати від правильності вибору цілей. Якщо не встановити пріоритети в масштабі всієї системи, окремі її частини можуть бути оптимізовані за різними параметрами (субоптимізація), в результаті чого система буде даремно витрачати ресурси. Навіть у невеликих проєктах програмісти часто забувають про справжні цілі і починають турбуватися про «ефективність» тоді, коли насправді вона не має принципового значення.

Встановлення пріоритетів для прийняття рішень найчастіше дає досить туманні результати. Не слід очікувати точних чисел, як то: швидкість 53 %, пам'ять 31 %, переносимість 15 %, вартість 1 %. Пріоритети рідко виявляються абсолютними. Якщо швидкість виконання важливіша, ніж пам'ять, це не означає, що будь-який приріст швидкості, навіть невеликий, може виправдати необмежене збільшення пам'яті, що витрачається. Не можна навіть скласти повний список критеріїв прийняття рішень. Пріоритети – це твердження філософії проєкту. Реальне прийняття рішень на подальших етапах все одно потребує оцінок та інтерпретації поставлених цілей.

5.2 Поширені архітектурні стилі

Серед існуючих систем широко поширені кілька архітектурних стилів, що можуть послужити вам як прототипи для побудови ваших додатків. Кожний стиль оптимальний для системи певного типу.

Коротко розглянемо такі різновиди стилів:

– пакетне перетворення (*batch transformation*) – одноразове перетворення всіх вхідних даних;

– безперервне перетворення (*continuous transformation*) – перетворення змінного вхідного сигналу, що виконується безперервно;

– інтерактивний інтерфейс (*interactive interface*) – система, в якій домінують зовнішні взаємодії;

– динамічне моделювання (*dynamic simulation*) – система, що моделює розвиваючі об'єкти реального світу;

– система реального часу (*real-time system*) – система, що відповідає жорстким тимчасовим обмеженням;

– адміністратор транзакцій (*transaction manager*) – система, що займається збереженням і оновленням даних, часто з підтримкою паралельного доступу з різних фізичних точок.

Звичайно, це не повний список відомих систем та архітектур. Для деяких задач потрібно придумувати абсолютно нову архітектуру, але в більшості випадків можна обійтися існуючим стилем або якоюсь з його модифікацій. Багато задач поєднують різні аспекти декількох архітектур.

Пакетне перетворення складається з декількох обчислювальних процесів. Програма отримує вхідні дані і має обчислити результат. Ніякої взаємодії з зовнішнім світом у проміжку не передбачається. Як приклади, можна привести стандартні обчислювальні завдання: компілятори, програми для розрахунку бухгалтерських відомостей, програми автоматичного проєктування та багато інших. Для таких завдань модель станів тривіальна або зовсім вироджена. Модель класів достатньо важлива: вона описує введення, виведення і проміжні стадії обчислень. Модель взаємодії документує обчислення та зв'язує між собою моделі класів. Найбільш важливим аспектом рішення задачі є визначення чіткої послідовності етапів.

Пакетне перетворення проєктується у наступній послідовності: розбити перетворення на етапи, кожен з яких полягає у виконанні частині перетворення; підготувати моделі класів для вводу, виводу та проміжних даних. Кожний етап

враховує тільки ті уявлення, що зв'язують його з сусідами; деталізувати кожний етап до тих пір, поки його реалізація не стане очевидною; оптимізувати отриманий конвеєр.

Безперервне перетворення – це характеристика системи, вихідний сигнал якої залежить від змінних сигналів на входах. На відміну від пакетного перетворення, яке обчислює результат один раз, безперервне перетворення повинно оновлювати вихідні сигнали досить часто (теоретично – безперервно, проте на практиці обчислення здійснюється з великою частотою дискретизації). Із-за жорстких тимчасових обмежень система не може обчислювати весь набір вихідних даних заново кожного разу при зміні вхідного сигналу (інакше це було б просто пакетне перетворення).

Замість цього система повинна розраховувати збільшення вихідних даних. В ролі типових прикладів програм цього класу можна навести: обробники сигналів, віконні системи, інкрементні компілятори та системи контролю виробничих процесів. Для цих систем моделі класів, станів і взаємодії повинні бути приблизно такими ж, як і для пакетних перетворень.

Безперервне перетворення проєктується у наступній послідовності: розбити перетворення на етапи, кожен з яких повинен описувати будь-яку частину перетворення; визначити моделі вхідних, вихідних і проміжних даних, як для пакетного перетворення; продиференціювати кожну операцію так, щоб працювати з прирощеннями; додати оптимізуючі проміжні об'єкти.

Інтерактивний інтерфейс **–** це система, в якій домінують взаємодії з зовнішніми агентами (людьми або пристроями). Зовнішні агенти не залежать від системи, тому вона не може їх контролювати, а може лише запитувати в них уведення даних. Інтерактивний інтерфейс звичайно становить лише частину деякого додатку, яка часто може розглядатися незалежно від обчислень. Як приклад інтерактивної системи можна навести:

інтерфейс запитів, заснований на формах, віконний менеджер робочої станції або панель управління моделюванням.

Найбільш важливі аспекти інтерактивного інтерфейсу – це протокол взаємодії системи з зовнішніми агентами, синтаксис можливих взаємодій, подання вихідних даних (варіанти відображення на екрані), потік керування всередині системи, продуктивність і обробка помилок. Інтерактивні інтерфейси визначаються головним чином моделлю станів. Модель класів описує елементи взаємодії (вхідні і вихідні маркери, формати подання). Модель взаємодії описує взаємодію діаграм станів.

Інтерактивний інтерфейс проєктується у такій послідовності: ізолювати класи інтерфейсу від класів додатку; використовувати зумовлені класи для опису взаємодії із зовнішніми агентами скрізь, де це можливо; використовуйте модель станів як структуру програми (інтерактивні інтерфейси найкраще реалізуються з використанням паралельного керування або подієвого керування); ізолюйте фізичні події від логічних; повністю опишіть функції програми, що викликаються інтерфейсом; переконайтеся, що в моделі є достатньо інформації для їх реалізації [5].

Динамічне моделювання (dynamic simulation) – це моделювання, або відстеження, об’єктів реального світу. Як приклади можна привести моделювання траєкторій руху молекул, розрахунок траєкторій космічних кораблів, економічні моделі та відеоігри. Моделювання найпростіше проєктувати з використанням об’єктно-орієнтованих технологій. Об’єкти та операції беруться безпосередньо з моделі додатку. Управління може бути реалізоване двома способами: зовнішній керівний об’єкт може імітувати кінцевий автомат, або ж об’єкти можуть обмінюватися повідомленнями між собою, як це відбувається в реальному світі.

На відміну від інтерактивної системи, внутрішні об’єкти при динамічному моделюванні відповідають об’єктам реального світу, тому модель класів у таких завданнях має істотне значення та

часто виходить досить складною. Моделі станів і взаємодії також важливі.

Динамічна модель проєктується у такій послідовності: ідентифікувати активні об'єкти реального світу в моделі класів, ці об'єкти мають періодично оновлені атрибути; ідентифікувати дискретні події; ідентифікувати безперервні залежності.

Зазвичай модель управляється циклом з невеликим кроком за часом. Дискретні події між об'єктами також часто включаються в цей цикл.

Зазвичай найскладніше при динамічному моделюванні – забезпечити адекватну продуктивність. В ідеальному світі довільна кількість паралельних процесорів могла б виконувати моделювання в точній відповідності з тим, як події розвиваються в реальному світі. На практиці проєктувальнику доводиться оцінювати обчислювальну вартість кожного циклу та забезпечувати систему достатніми обчислювальними ресурсами. Безперервні процеси доводиться апроксимувати дискретними шагами.

Система реального часу – це інтерактивна система з жорсткими тимчасовими обмеженнями на час відгуку. Системи реального часу бувають жорсткими та гнучкими. Жорсткі – це життєво важливі додатки, що вимагають гарантованого відгуку в заданий час. Гнучкі системи реального часу теж повинні бути високонадійними, але для них допустиме рідкісне порушення обмежень. Типові додатки реального часу – це управління процесами, зчитування даних, комунікаційні пристрої, керуючі пристрої та ін.

Проєктування систем реального часу – складне завдання, що вимагає вирішення таких підзавдань, як оброблення переривань, розподіл за пріоритетами та координація роботи безлічі процесорів [6]. На жаль, системи реального часу часто використовуються в режимах, близьких до гранично припустимих, тому для досягнення необхідної продуктивності

доводиться змінювати проєкт системи. У результаті втрачається переносимість та зручність обслуговування системи.

Адміністратор транзакцій – це система, основне призначення якої полягає в зберіганні та видачі даних. Більшість адміністраторів транзакцій мають справу з безліччю користувачів, що одночасно зчитують і записують дані. Адміністратор повинен забезпечувати захист даних від несанкціонованого доступу і від випадкових збоїв. Адміністратори транзакцій часто реалізуються у вигляді надбудови над системою управління базами даних (СУБД). СУБД надає універсальну функціональність для управління даними, яка може бути використана повторно. Як приклади адміністраторів транзакцій можна навести системи бронювання авіаквитків, контролю інвентарю та виконання замовлень.

У таких системах домінує модель класів. Модель станів буває важливою в окремих випадках, зокрема для опису еволюції об'єкта, а також обмежень і методів, що застосовуються в різні моменти часу. Модель взаємодії буває суттєвою досить рідко.

Адміністратор транзакцій проєктується у наступній послідовності: перетворити модель класів до структур бази даних; визначити паралельні модулі – такі, які не можуть використовуватися спільно; визначити одиницю транзакції – набір ресурсів, що одночасно задіяні в одній транзакції; виконати проєктування паралельного управління транзакціями.

5.3 Шаблони проєктування медичної інформаційної інфраструктури

Шаблон або патерн проєктування медично інформаційних систем - це загальне рішення, що підходить для повторного використання, при виникненні проблем проєктування систем [7]. Патерн не є готовим рішенням, яке можна просто додати в проєктовану систему і вирішити проблему. Він є повторюваним архітектурним

рішенням, яке описує підхід до вирішення проблеми і застосовується в різних ситуаціях. Шаблони бувають трьох видів: породжувані, структурні і поведінкові (шаблони поведінки) [8-14].

5.4 Породжувані шаблони

Стандартний спосіб створення об'єктів або класів може привести до проблем в архітектурі або до її ускладнення. Породжувані шаблони намагаються усунути ці проблеми, використовуючи механізми створення об'єктів і класів, які забезпечують зручне і безпечне створення нових об'єктів і класів, у відповідних ситуаціях. Вони дозволяють зробити систему незалежною від способу створення, композиції та представлення об'єктів [7].

Породжувані шаблони використовуються, коли інформаційна система більше залежить від композиції об'єктів, ніж від успадкування класів. вони інкапсулюють знання про певні класах і приховують деталі того, як створюються і з'єднуються класи. Породжувані патерни надають додаткову гнучкість при відповідях на питання: «Що і як створюється?» і «Хто і коли створює?

Ці шаблони виходять на перший план, коли успіх програмного продукту в більшій мірі залежить від композиції інформаційних об'єктів, ніж від її класів. Основний акцент життєдіяльності таких систем робиться на визначення обмеженого набору фундаментальних функцій, за допомогою поєднання яких можна отримувати більш складні функції.

Патерни даної групи приховують в собі інформацію про класи, які застосовуються в системі, деталі механізмів створення і взаємодії цих класів.

Єдина інформація про об'єкти, яка повинна бути відома, це інтерфейси, визначені за допомогою абстрактних класів.

Це призводить до того, що породжують шаблони забезпечують більшу гнучкість при вирішенні питання про те, що

створюється, хто це створює, як і коли. Можна зібрати систему з "готових" об'єктів з самої різною структурою і функціональністю статично (на етапі компіляції) або динамічно (під час виконання).

Дуже часто виникають ситуації, коли доцільно використовувати кілька шаблонів одночасно для вирішення одного завдання. У таких випадках вони доповнюють один одного (застосовуючи шаблон проєктування "Будівельник", можна використовувати і інші шаблони для вирішення завдання реалізації його компонентів).

5.4.1 Шаблон «Абстрактна фабрика»

В умовах, коли потрібно створити сімейство взаємопов'язаних або взаємозалежних об'єктів, не визначаючи при цьому для них конкретних класів, найбільш очевидним рішенням буде реалізація абстрактного класу, в якому повинен бути оголошений інтерфейс для створення конкретних класів.

Шаблон, який описує рішення задачі створення об'єктів шляхом їх ініціалізації в класі загального призначення, носить назву "Абстрактна фабрика".

Необхідні рішення досягаються шляхом координації процесу створення об'єктів, в якому крім конкретних інтерфейсів повинні бути визначені правила створення тих чи інших сімейств об'єктів.

Основними недоліками цього шаблону є те, що інтерфейс абстрактної фабрики фіксує набір об'єктів, які можна створити. Подальша модернізація, припускає включення нових об'єктів, - часто скрутна і ресурсномістка операція.

До переваг можна віднести, що шаблон допомагає контролювати класи об'єктів, що створюються додатком.

Клас "Конкретної фабрики" з'являється в додатку тільки один раз за інстанціювання. Це полегшує заміну використовуваної додатком "Конкретної фабрики". Додаток може змінити конфігурацію продуктів, просто підставивши нову "Конкретну

фабрику". Оскільки "Абстрактна фабрика" створює все сімейство продуктів, то і замінюється відразу все сімейство.

Якщо продукти деякого сімейства спроєктовані для спільного використання, то важливо, щоб додаток в кожен момент часу працював тільки з продуктами єдиного сімейства.

Шаблон "Абстрактна фабрика" застосовується, коли необхідно координувати створення інформаційних об'єктів. Головною його якістю є те, що він дозволяє відокремити реалізацію створення екземплярів об'єктів від об'єкта, який буде використовувати ці екземпляри.

5.4.2 Шаблон «Одинак»

Основний сенс використання патерну «Одинак» складається в реалізації єдиного спеціального класу, до якого повинні звертатися інші інформаційні об'єкти, через єдину точку доступу. Це вирішується за допомогою створення класу і визначення статичних методів цього класу, які повертають цей єдиний об'єкт. Застосування цього шаблону буде гарантувати, що у класу з'явиться тільки один екземпляр, і зможе надати до нього точку доступу [7].

При реалізації цього шаблону розумніше створювати саме статичні екземпляри абстрактного класу, а не оголошувати необхідні методи статичними. Це буде додатковою перевагою використання цього шаблону, оскільки при застосуванні методів екземпляра можна використовувати механізм успадкування і створювати підкласи.

Рішення на основі створення екземпляра є більш гнучким, оскільки згодом може знадобитися вже не один екземпляр об'єкта, а кілька.

В якості основних переваг використання цього шаблону проєктування можна виділити наступні:

– контрольований доступ до єдиного екземпляру.

– шаблон допускає уточнення операцій над екземплярами і їх уявлення.

– висока гнучкість при модифікації класу.

Єдиним його недоліком є те, що в деяких випадках він призводить до створення немасштабуємого додатку.

5.4.3 Шаблон «Прототип»

Будь-яка сучасна інформаційна система не повинна залежати від того, як в ній створюються, компонуються і представляються об'єкти.

Для того щоб практично підтримати даний постулат, слід створювати нові об'єкти за допомогою патерна "Прототип". Цей шаблон оголошує інтерфейс для клонування самого себе. Логіка додатка, створюючи новий об'єкт, звертається до "прототипів" із запитом його клонування [8].

Реалізація цього шаблону в окремих деталях нагадує патерн "Абстрактна фабрика" .Він також служить для створення об'єктів, однак з іншим підходом.

Відмінними умовами, що характеризують доречність застосування шаблону "Прототип", є:

– визначення створюваного об'єкта динамічно, під час виконання;

– небажане створення окремої ієрархії класів фабрик для створення об'єктів-продуктів з паралельної ієрархії класів (відміну від шаблону "Абстрактна фабрика");

– функціонал клонування об'єкта є кращим варіантом, ніж його створення і ініціалізація за допомогою конструктора;

– особливо в ситуаціях, коли об'єкт може приймати невелику обмежене число можливих станів.

Потрібно зауважити, що клонування не обов'язково повинно проводитися в реалізації самого шаблону "Прототип" - це може

бути і якийсь інший об'єкт, до якого сам "Прототип" повинен мати постійний і повний доступ.

Аналогією цього шаблону є шаблон "Шаблоновий метод". Різниця полягає в тому, що "Шаблоновий метод" - поведінковий шаблон проєктування, що визначає основу алгоритму, на якій екземпляри об'єктів можуть перевизначати окремі кроки загального алгоритму, а "Прототип" - породжує шаблон, на основі якого об'єкти створюються [8].

5.4.4. Шаблон «Творець примірників»

В умовах, коли не визначено, який з компонентів системи повинен відповідати за створення екземплярів класу, використовують патерн "Творець примірників" [8].

Ідея реалізації цього шаблону полягає в тому, що відповідальність за створення об'єктів класу 1 делегується класу 2.

Передумови до створення подібного шаблону проєктування виходять з необхідності класу 2 агрегувати, використовувати, модифікувати об'єкти класу 1. Типовим структурним об'єктом, створеним на основі патерну "Творець примірників", є розглянутий нами раніше шаблон "Інформаційний експерт".

Основною перевагою використання даного патерну є збереження гнучкості системи через те, що створені об'єкти класу 1, як правило, видно тільки для класу 1. Але в ситуації, коли процедура створення класів є складною і багатоетапної, логічніше буде використовувати шаблон проєктування "Абстрактна фабрика".

5.4.5 Шаблон «Будівельник»

Якщо потрібно відокремити конструювання складного об'єкта від його уявлення таким чином, щоб в результаті одного і того ж конструювання могли виходити різні уявлення, використовують

шаблон "Будівельник". Суть цього шаблону полягає в тому, що алгоритм створення складного об'єкта не повинен залежати від того, з яких частин складається об'єкт і як ці частини повинні взаємодіяти між собою.

Шаблон "Будівельник" ізолює код, який реалізує створення об'єкта і його уявлення.

Для того щоб реалізація "Абстрактної фабрики" була виконана, необхідно дотримуватися наступних рекомендацій:

– однозначна ідентифікація правила створення екземплярів;

– визначення абстрактного класу з інтерфейсом, що включає окремий метод для кожного з класів, екземпляри яких повинні бути створені;

– для кожного сімейства необхідно створювати конкретні класи, похідні відданого абстрактного класу;

– при використанні екземпляру об'єкту необхідно звертатися до "Абстрактної фабрики" для створення необхідних примірників.

5.5 Структурні шаблони

В якості структурних проєктування виділяють наступні: "Адаптер", "Декоратор", "Заступник", "Інформаційний експерт", "Компонувальник", "Міст", "Низька зв'язаність", "Пристосуванець", "Стійкий до змін", "Фасад" [9].

Деякі структурні шаблони часто використовуються спільно, вирішуючи спеціалізовані практичні завдання, але кожну задачу можна вирішити, використовуючи різні патерни проєктування. Різниця у виборі того чи іншого патерна повинна бути обґрунтована з точки зору вибору конкретних умов використання застосунку, розробленого із застосуванням обговорюваних шаблонів. Шаблон повинен знімати проблеми, що виникли тут і зараз. При цьому далекоглядність розробника полягає в тому, наскільки просто запропоноване шаблонне рішення зможе підтримати майбутні програмні зміни і перетворення.

5.5.1 Шаблон «Адаптер»

Структурний шаблон "Адаптер" використовують в ситуації, коли необхідно організувати використання функцій певного бізнес-об'єкта, недоступного для модифікації. Це відбувається, коли система або її компонент мають необхідні дані і функціональність, але не мають відповідного інтерфейсу [10].

Якщо сформулювати його призначення інакше, то шаблон "Адаптер" забезпечує взаємодію несумісних інтерфейсів шляхом надання єдиного стійкого інтерфейсу для декількох компонентів.

Алгоритм реалізації шаблону "Адаптер" заснований на конвертації від вихідного інтерфейсу конкретного компонента до іншого виду за допомогою проміжного об'єкта-безпосередньо самого "Адаптера" .Тобто розробляється спеціальний об'єкт із загальним інтерфейсом, який перенаправляє зв'язку від зовнішніх об'єктів до самого "Адаптеру". Цей структурний шаблон є своєрідним втіленням принципу об'єктно-орієнтованого програмування - поліморфізму. Система містить об'єкти різних типів, але використовують їх об'єкти взаємодіють одним і тим же чином, використовуючи "Адаптер" для перетворення різних даних до єдиної поданням.

Найчастіше "Адаптер" застосовується, якщо необхідно створити певний клас, похідний від вже існуючого. Іншими словами, даний шаблон передбачає створення нового інтерфейсу для необхідного об'єкта, який ми не можемо використовувати через його невідповідного інтерфейсу. Це шаблон загального характеру і дуже часто використовується в комбінації з іншими шаблонами проєктування.

"Перетворення вихідного інтерфейсу класу в інтерфейс, більш відповідний для конкретних потреб. Застосування цього шаблону дозволяє організувати спільну роботу несумісних інтерфейсів".

Шаблон "Адаптер" дозволяє включати вже існуючі об'єкти в нові структури, незалежно від відмінностей в їх інтерфейсах.

Інтерфейс включає об'єкта і його клас приводиться у відповідність з новими вимогами, а виклики методів перетворюються в виклики методів включеного класу. Застосування шаблону "Адаптер" під час розробки програмного продукту дозволяє не брати до уваги можливі відмінності в інтерфейсах вже існуючих об'єктів. Якщо є клас, що володіє необхідними методами і властивостями, то при необхідності завжди можна буде скористатися даним шаблоном для приведення його інтерфейсів до потрібного вигляду. Це особливо важливо, якщо застосовується відразу кілька шаблонів, оскільки багато шаблони вимагають, щоб використовувані в них класи були породжені від одного і того ж класу. На сьогоднішній момент поширені два типи цього шаблону:

1. Об'єктний. Варіант шаблону, який реалізується за допомогою приміщення, що адаптується в інший, що адаптує об'єкт.

2. Класовий. Використовує механізм множинного спадкоємства і отримав назву класового.

5.5.2 Шаблон «Декоратор»

Якщо необхідно покласти додаткові обов'язки на окремий об'єкт, а не на клас, що генерує об'єкти, доцільно застосовувати шаблон "Декоратор"[10].

Можливість динамічно додавати нові обов'язки, не вдаючись до породження підкласів, полегшує розуміння основних обов'язків об'єктів і не заплутує основну бізнес-логіку інформаційних систем:

– системні компоненти визначають інтерфейс об'єкта, на який можуть бути динамічно покладені додаткові обов'язки;

– конкретні компоненти визначають об'єкт, на який покладаються додаткові обов'язки;

– шаблон "Декоратор" зберігає посилання на об'єкт (компонент), визначає його інтерфейс і переадресує на нього робочі запити.

"Декоратор" дозволяє реалізувати більшу гнучкість, ніж у статичного успадкування, що створюється шляхом розробки підкласів.

Можна додавати і видаляти обов'язки під час виконання програми, в той час як при використанні успадкування треба було б створювати новий клас для кожного додаткового обов'язку.

Даний шаблон дозволяє не створювати класи, перевантажені методами. Нові обов'язки можна додавати тільки в міру необхідності, не перевантажуючи програмне забезпечення зайвими класами.

5.5.3 Шаблон «Заступник»

У випадках, коли необхідно керувати доступом до об'єкту, так щоб створювати громіздкі компоненти тільки "на вимогу", оптимально використовувати шаблон "Заступник".

Суть "Заступника" полягає в тому, що він зберігає посилання, яке дозволяє йому звернутися до реального суб'єкту тільки при необхідності, задіюючи системні ресурси.

Об'єкти, які створюються класами, реалізованими на основі системного компоненту, якщо їх інтерфейси ідентичні, оскільки інтерфейс компонента повинен бути ідентичний інтерфейсу заступника так, що вони можуть бути взаємозамінні і заступник може контролювати створення, зміна, видалення компонента або його примірника.

Шаблон "Заступник" може мати й інші обов'язки:

– віддалений "Заступник" може відповідати за кодування запиту і його аргументів і відправку закодованого запиту;

– віртуальний "Заступник" може кешувати додаткову інформацію про реальний компоненті;

– захищає "Заступник" може перевіряти, чи має викликає об'єкт необхідні для виконання запиту права;

– шаблон "Заступник" досить часто використовують в парі з шаблоном "Адаптер".

5.5.4 Шаблон «Інформаційний експерт»

Коли в системі повинна акумулюватися, перетворюватися і віддалятися необхідна інформація, застосовується шаблон "Інформаційний експерт".

Сенс реалізації цього патерну полягає в тому, що відповідальність за акумуляцію і перетворення інформації призначається певній системному класу, який повинен володіти необхідною інформацією і функціональністю.

Роль інформаційного експерта може грати не тільки один, але і кілька спеціалізованих класів, кожен з яких повинен відповідати за свої функціональні можливості. Обов'язки розподіляються відповідно до наданої за результатами виконання інформацією. Таким чином підтримується інкапсуляція, тобто об'єкти використовують свої власні дані для виконання поставлених завдань.

Якщо об'єкт, що володіє найбільш повною інформацією, буде відповідати і за збереження цієї інформації в базі даних, то вийде, що логіка програми та логіка зв'язку з базою даних "поміщаються" в один клас, що порушує принцип поділу обов'язків основних об'єктів системи, і, крім того, логіка зв'язку з базою даних буде дублюватися в багатьох інших класах.

5.5.5 Шаблон «Компонувальник»

Застосування шаблону "Компонувальник" особливо доречне, коли в інформаційній системі реалізовані і підтримуються деревовидні структури об'єктів. Яскравим прикладом типової

деревовидної структури є призначений для користувача інтерфейс [11]: “Вікно призначеного для користувача інтерфейсу - контейнер для більш простих елементів - панелей, кнопок, полів введення” і т.д.

Для вирішення завдання створення і обробки деревовидних елементів необхідно завести загальний інтерфейс, який буде описувати і елементарні, і складові об'єкти.

Так як інтерфейс описує і складові об'єкти, він повинен містити контейнерні методи для додавання, видалення і отримання об'єкта з контейнера. Причому функціональні методи повинні бути параметризовані загальним інтерфейсом. Організація подібного структурного шаблону призводить до можливості автоматичного додавання в контейнер не тільки елементарних об'єктів, а й інших контейнерів.

Перевагами шаблону "Компонувальник" є:

– легкість додавання нових примітивних або складових об'єктів;

– простота структури програми:

– примітивні і складові елементи обробляються однаковим чином.

До недоліків слід віднести незручність реалізації заборони на додавання в складений об'єкт компонентів певних типів.

5.5.6 Шаблон «Міст»

"Міст" - один з найважчих для розуміння шаблонів проєктування. Це викликано тим, що він є дуже потужним і часто застосовується в самих різних ситуаціях. Крім того, він суперечить поширеній практиці застосування механізму наслідування для реалізації спеціальних випадків.

Алгоритм його реалізації полягає в приміщенні абстракції і реалізації в окремі ієрархії класів.

Застосування шаблону "Міст" дозволяє отримати більш стійкі проєктні рішення для представлення елементів абстракції і реалізації, спрощуючи їх можлива подальша зміна.

5.5.7 Шаблон "Низька зв'язаність"

Шаблон "Низька зв'язаність" втілює принцип проєктування, який дозволяє розподілити обов'язки між об'єктами таким чином, щоб ступінь пов'язаності між системами залишалася низькою.

Ступінь зв'язаності - міра, яка визначає, наскільки жорстко один елемент пов'язаний з іншими елементами, або якою кількістю даних про інших елементах він володіє.

Отже, розглянутий патерн використовується, щоб забезпечити низьку зв'язаність при створенні системних примірників і зв'язуванні їх з іншими. Суть шаблону "Низька зв'язаність" полягає в розподілі обов'язків між об'єктами так, щоб ступінь пов'язаності залишалася на низькому рівні.

5.5.8 Шаблон «Пристосуванець»

Коли необхідно забезпечити підтримку безлічі дрібних об'єктів, ініціалізованих і використовуваних в інформаційній системі, застосовують структурний шаблон проєктування - "Пристосуванець" [10].

"Пристосуванець" - це екземпляр об'єкта, який видає себе за групу самостійних примірників.

Реалізація шаблону виконується за рахунок створення розділяється об'єкта, який можна використовувати одночасно в декількох контекстах. У кожному контексті він повинен виглядати як незалежний об'єкт, тобто бути не відрізнятись від вихідного примірника. Патерн "Пристосуванець" декларує інтерфейс, за допомогою якого конкретні пристосуванці можуть отримати зовнішній стан або якось впливати на нього. Крім іншого,

екземпляр пристосуванця зберігає внутрішній стан, в той час як зовнішній стан зберігається або обчислюється використовують його процесами обробки.

Наявність різних варіантів внутрішнього стану призводить до необхідності створення набору пристосуванців. При запиті нового екземпляра виконується пошук вже реалізованого варіанта. Якщо такого не знаходиться, то породжується новий.

5.5.9 Шаблон «Фасад»

Призначення шаблону "Фасад" - спростити роботу з існуючою системою, визначивши власний інтерфейс звернення до неї. При цьому визначається інтерфейс повинен використовувати тільки певну безліч функцій системи або організувати взаємодію з нею якимось специфічним чином.

Реалізація цього шаблону визначає точку взаємодії з підсистемою - фасадний об'єкт, що забезпечує загальний інтерфейс, який взаємодіє з компонентами.

В основі реалізації шаблону лежить визначення нового класу (або класів) з необхідним інтерфейсом, які повинні спиратися на функціональні можливості існуючої системи.

"Фасад" може застосовуватися не тільки для створення спрощеного інтерфейсу з метою виклику методів, але і для зменшення кількості об'єктів, з якими клієнтові доводиться взаємодіяти.

Цей шаблон також може використовуватися для приховування або інкапсуляції базової системи. В цьому випадку профільні класи включають основну систему як свій закритий компонент. Тому основна система буде взаємодіяти тільки з "Фасадом", залишаючись недоступною і невидимою для всіх інших користувачів.

5.6 Шаблони поведінки

У поведінкових шаблонах, як і в суміжних їм структурних шаблонах, використовується спадкування, як інструмент визначення поведінки для різних класів, і композиція, як зрівнювач і розподільник виконуваних ними обов'язків.

5.6.1 Інтерпретатор

"Інтерпретатор" - поведінковий шаблон проєктування, який вирішує завдання, які часто зустрічаються, але піддаються змінам. Також відомий як Little (Small) Language.

Для його розробки потрібні наступні учасники:

– абстрактний вираз;

– визначення інтерфейсу вираження, оголошення методу шаблону;

– термінальний вираз;

– для кожного об'єкта створюється свій термінальний вираз;

– нетермінальних вираз;

– для кожного окремого правила створюється свій об'єкт нетермінального вираження;

– контекст;

– клієнт;

– управління виконанням бізнес-логіки у вигляді абстрактного синтаксичного дерева, вузлами якого є об'єкти термінального і нетермінального вираження;

– методи "інтерпретатора" в нетермінальних виразах дозволяють реалізувати правила. При цьому ми легко можемо додати нові правила, визначивши нові об'єкти нетермінального вираження зі своєю реалізацією методу шаблону.

Однак недолік даного шаблону полягає в тому, що він підходить тільки для тих випадків, коли правила відносно прості і не містять великої кількості відгалужень. У більш складних випадках слід вибирати інші способи проєктування програми.

Шаблони поведінки вирішують завдання, позначені вище, і оптимізують загальну продуктивність системи.

5.6.2 Ітератор

У випадках, коли потрібно, щоб складний складений об'єкт, наприклад список, надавав доступ до своїх елементів (об'єктів), не розкриваючи їх внутрішню структуру, причому перебирати список потрібно по-різному в залежності від завдання, застосовується шаблон "Ітератор" [12].

Реалізація шаблону повинна мати абстракцію, що дозволяє розділити класи колекцій і алгоритмів.

Таким чином, даний патерн використовується, коли необхідний механізм "абстрактного" обходу різних структур даних так, щоб були визначені алгоритми, здатні взаємодіяти зі структурами прозоро. Якщо розкрити тему реалізації та можливостей цього шаблону, то слід сказати про те, що будь-який складений об'єкт, такий як список, повинен надавати спосіб доступу до його елементів без розкриття своєї внутрішньої структури. Іноді це є не просто варіантом використання структури даних, а потрібно перебирати елементи списку різними способами, в залежності від конкретного завдання, а іноді потрібно мати кілька активних обходів одного списку одночасно. Ідеальним рішенням в такому випадку є єдиний інтерфейс для обходу різних типів складових об'єктів.

Алгоритм реалізації даного шаблону наказує наступні стадії:

– створюється певний клас ("Ітератор"), який визначає інтерфейс для доступу і перебору елементів;

– конкретний екземпляр класу "Ітератор" реалізує його інтерфейс і стежить за поточною позицією при обході агрегату;

– агрегат визначає інтерфейс для створення об'єкта-ітератора;

– конкретний екземпляр агрегату реалізує інтерфейс створення ітератора і повертає екземпляр його класу;

– конкретний ітератор відстежує поточний об'єкт в агрегаті і може обчислити наступний об'єкт при переборі.

5.6.3 Команда (Транзакція)

Коли необхідно надіслати об'єкту запит, не знаючи про те, виконання якої операції запрошено і хто буде одержувачем, доцільно застосовувати шаблон "Команда".

Основна ідея даного шаблону полягає в використанні єдиного інтерфейсу для опису всіх типів операцій, які можна виробляти з системою. Для додавання в систему підтримки нової операції досить реалізувати необхідний інтерфейс. Кожна операція представляється самостійним об'єктом, яка інкапсулює деякий набір додаткових властивостей.

У цьому шаблоні алгоритм представлення бізнес-логіки організований у вигляді послідовності процедур, які керують кожна своїм запитом. Будь-який додаток можна представити у вигляді набору транзакцій. Якісь із них вибирають дані, якісь - змінюють. Кожна взаємодія користувача і системи містить певний набір дій. Патерн "Команда" організовує всю використовувану логіку в одну "над"-процедури, працюючи зданими безпосередньо або через тонку обгортку.

5.6.4. Спостерігач

У випадку, коли інформаційна система складається з безлічі різних класів і взаємодіючих об'єктів, які знаходяться в узгоджених станах між собою, коли необхідно зберігати гнучкість системи або уникати монолітності шляхом підвищення спроможності перевикористання існуючих класів, прийнято використовувати шаблон проєктування "Спостерігач" [13].

Таким чином, цей шаблон визначає залежність "один-до-багатьох" між об'єктами так, що при зміні стану одного об'єкта всіх залежних від нього об'єкти повідомляються і оновлюються автоматично.

Об'єкт представляє основну, досить незалежну абстракцію, "спостерігач" є залежною абстракцією. Об'єкт сповіщає спостерігачів про своє зміні, на що кожен спостерігач буде реагувати по-своєму, може запитати додаткову інформацію від об'єкта, а може проігнорувати змінилося стан.

Основні переваги шаблону "Спостерігач" :

– мінімальна зв'язаність об'єкта і спостерігача;
– об'єкт знає лише про те, що у нього є ряд спостерігачів;
– широкомовність оповіщення;
– об'єкт оповіщає не конкретного, а всіх підписаних на нього спостерігачів.

Недоліками є:

– непередбачені поновлення;
– зміна об'єкта може викликати каскад залежних від нього спостерігачів з високою вартістю поновлення;
– протокол оновлення не містить ніяких відомостей про те, що змінилося в суб'єкті, - робота спостерігачів при цьому ускладнюється.

5.6.5 Стратегія

Коли певний клас містить ряд схожих алгоритмів (способів зробити ту чи іншу дію), як правило, ці алгоритми призводять до одного і того ж результату, але можуть відрізнятися за іншими параметрами (час виконання, споживання системних ресурсів і ін.). У подібних ситуаціях доцільно використовувати шаблон "Стратегія".

В основній ідеї даного шаблону лежить необхідність виділення сімейства схожих алгоритмів і інкапсуляції кожного з

них в власний клас. Після цього алгоритми можна замінювати прямо під час виконання програми.

5.6.6 Зберігач

Коли необхідно зафіксувати поведінку об'єкта для його подальшої реалізації, застосовується шаблон " Зберігач".

Цей шаблон застосовується, якщо потрібно зафіксувати і винести (не порушуючи структури програми) за межі об'єкта його внутрішній стан так, щоб згодом можна було відновити в ньому об'єкт [12,14].

Таким чином, не порушуючи принципу інкапсуляції, шаблон " Зберігач" отримує і зберігає за межами об'єкта його внутрішній стан так, щоб пізніше можна було відновити об'єкт в первісному стані.

Особливо актуально застосування цього патерну в ситуації, коли потрібно відновити об'єкт назад до свого попереднього стану. Шаблон проєктування " Зберігач" фіксує і зберігає за межами об'єкта " Зберігач " його внутрішній стан так, щоб пізніше цей об'єкт можна було б відновити в такому ж стані.

Цей шаблон використовується в функціональності тих додатків, коли є можливість скасувати останню дію і повернути систему в попередній стан.

5.6.7 Ланцюжок обов'язків

У випадках, коли потрібно ефективно, компактно, надійно реалізувати обробку потоку інформації з потенційно великою кількістю обробників, використовується шаблон проєктування "Ланцюжок обов'язків".

У сучасних системах кількість обробників багато в чому визначає оперативність виконання системою її функцій, тому їх кількість, як правило, досить велике.

Патерн "Ланцюжок обов'язків" дозволяє реалізувати ефективний механізм ув'язування типів повідомлень і обробників [14].

Якщо в системі невелике число обробників, то досить реалізувати даний механізм на базі циклової конструкції.

Цей шаблон проєктування досить вимогливий до системних ресурсів. Зокрема, необхідно переконатися, що програма коректно "відловлює" випадки необроблених запитів.

Оптимальне використання поведінкових шаблонів дозволить нівелювати недоліки невдало спроєктованого програмного забезпечення, знизити окремі, найбільш негативні або підвищити вдалі характеристики програмного продукту.

Саме високий рівень володіння деталями реалізації поведінкових патернів допоможе розробнику інформаційної системи оперативно усунути найбільш значущі недоліки окремих компонентів інформаційної системи.

.

5.7 Практичні завдання

Варіант 1. Розробіть архітектуру системи, яка буде обробляти дані з датчика для виміру температури тіла, артеріального тиску (тонометр), датчика для виміру пульсу та зберігати у хмарному сховищі. Система повинна мати як веб інтерфейс так і окреме мобільне застосування.

Варіант 2. Розробіть архітектуру системи, яка буде обробляти сигнали з датчика для виміру рівня кисню в крові (SPO2 та зберігати у хмарному сховищі. Система повинна мати як веб інтерфейс так і окреме мобільне застосування.

Варіант 3. Розробіть архітектуру системи, яка буде обробляти сигнали з датчика для виміру повітряного потоку (дихання) і поміщати та зберігати у хмарному сховищі. Система повинна мати як веб інтерфейс так і окреме мобільне застосування.

Варіант 4. Розробіть архітектуру системи,, яка буде обробляти сигнали з датчика для виміру ЕКГ (ECG) та зберігати у хмарному сховищі. Система повинна мати як веб інтерфейс так і окреме мобільне застосування.

5.8 Контрольні запитання

1. З яких етапів складається створення архітектури системи?

2. У чому різниця між бібліотеками та каркасами?

3. Якими якостями повинні володіти «хороші» бібліотеки класів?

4. Що розуміють під терміном шаблон?

5. У чому різниця між шаблонами та каркасами?

6. Дайте визначення підсистеми.

7. Як можуть бути організовані відносини між підсистемами?

8. Як організована багатошарова система?

9. У чому різниця розділів та шарів при організації системи?

10. Який засіб є основним при виділені паралелізмів?

11.Які кроки необхідно виконати при розподілі підсистем?

12.Які типи даних краще зберігати у файлах, а які в базах даних? Наведіть приклад.

13. Які глобальні ресурси повинен виділити проєктувальник?

14. Які варіанти управління програмною системою існують?

15. Яким граничним умовам проєктувальник повинний приділити увагу?

16. Назвіть існуючі архітектурні стилі. Дайте їм визначення та коротку характеристику.

17. Дайте визначення компонента. В чом його відмінність від класу та пакету?

18. Які властивості мають компоненти?

19. Які основні категорії інтерфейсів?

20. Назвіть характеристики інтерфейсів.

21.Які стандартні компонентні моделі вам відомі? Дайте їх визначення та коротку характеристику.

22. Наведіть класифікацію шаблонів проєктування.

5.9 Література до розділу

1. P. Arras, Architectural Characteristics and Educational Possibilities of the Remote Laboratory in Materials Properties/ P. Arras, G. Tabunshchyk, Ye. Kolot, B. Tanghe// REV2014 Conference26 – 28 February 2014, Porto, Portugal, PP. 94-97

2. Remote and virtual tools in engineering: student textbook /general editorship Dr.Ing.Karsten Henke. – Zaporizhzhya: Dike Pole, 2016. – pp. 250.

3. Tabunshchyk G. Multipurpose Educational System based on Raspberry Pi/ G. Tabunshchyk, D. Van Merode , O.Petrova, V. Okhmak // Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering, Nitra, Slovakia, 12-15 September, 2016 – pp. 202-206

4. Ohmak V.A. Conveyor Model for Embedded Systems Study /V.A. Ohmak, G.V Tabunshik// Proceedings of the International Symposium on Embedded Systems and Trends in Teaching Engineering, Nitra, Slovakia, 12-15 September, 2016 –pp.60-64.

5. Merode D. Van Merode Interactive university platform // Merode D. Van, Tabunshchyk G., Goncharov Y., Patrakhalko K., Staroverov V.// Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : тези доповідей VIII Міжнародної науково-практичної конференції (21–23 вересня 2016 р., м. Запоріжжя). – Запоріжжя : ЗНТУ, 2016. – 344 с.

6. M. Fowler Refactoring: Improving the Design of Existing Code / M. Fowler,K. Beck,J. Brant,W. Opdyke,D. Roberts. – USA: ADDISON-WESLEY, 2012. – 455 p.

7. Эванс Э. Предметно-ориентированное проєктирование (DDD): структуризация сложных программных систем. [Текст] / Эванс Э. – М.:Вильямс, 2010. – 448 с.

8. Фримен Э. Паттерны проектирования. / Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс // Питер, – 2017. – 656 с.

9. Мисюра, М. А. Шаблоны проектирования GoF. Структурные шаблоны. Адаптер и декоратор / М. А. Мисюра. // Молодой ученый. — 2017. — № 4 (138). — С. 172-174.

10. Зайнетдинов, А. Р. Использование шаблонов проектирования информационных систем / А. Р. Зайнетдинов, А. В. Недяк, О. Ю. Рудзейт // Молодой ученый. — 2019. — № 34 (272). — С. 14-18.

11. Лясин, Д. Н. Реализация архитектурного шаблона MVC с использованием шаблона проектирования «Наблюдатель» на языке PHP / Д. Н. Лясин, О. Н. Симонова. // Молодой ученый. — 2014. — № 6 (65). — С. 108-111.

12. Тепляков С. Паттерны проектирования на платформе .NET. / С. Тепляков., 2015. – 320 с.

13. Паттерны проектирования / Э.Фримен, Б. Бейтс, К. Сьерра, Э. Фримен., 2017. – 656 с.

14. Гамма Э. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Джонсон, Р. Хелм., 2020. – 448 с.

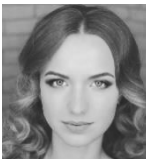
АВТОРИ

ТАБУНЩИК Галина Володимирівна – к.т.н., професор, професор Національного університету «Запорізька політехніка». Закінчила Запорізький державний технічний університет за спеціальність програмне забезпечення автоматизованих систем, захистила дисертацію на звання кандидат технічних наук зі спеціальності 05.13.03 – системи і процеси керування.

Має більш ніж 150 наукових праць. Наукові інтереси – інженерія програмного забезпечення, верифікація інформаційних систем, програмування вбудованих систем, керування ризиками.

ORCID ID: 0000-0003-1429-5180

Galyna TABUNSHCHYK– PhD, Prof of Software Tool Department of National Univerity Zaporizhzhia Polytechnic. Graduated from Zaporizhzhya National Technical University with speciality Software Engineering, in 2004 finished PhD work in control systems and process. Have more than 100 scientific works. Scientific interests – Software Engineering, System Verification, Embeded Systems, Risk Management



КАПЛІЄНКО Тетяна Ігорівна – кандидат технічних наук, доцент кафедри програмних засобів Запорізького національного технічного університету. Закінчила магістратуру за спеціальністю «Програмне забезпечення автоматизованих систем», аспірантуру за спеціальністю «Інформаційні технології».

Автор 33 наукових публікацій, 3 авторських свідоцтв і одного патенту. Основні напрямки наукових досліджень: аналіз та верифікація якості програмного забезпечення; керування програмними проєктами; керування ризиками.

ORCID ID: 0000-0001-8192-2397

Tetiana KAPLIIENKO, Ph.D., an associate professor of Software Tools Department at Zaporizhzhya National Technical University. She has a master's degree in "Software Engineering". She finished PhD study in "Information technology". She is the author of 33 scientific publications, 3 Certificates for invention and 1 patent. Main research fields: the analysis and verification for the software quality; software design managements; risks management

ПЕТРОВА Ольга Анатоліївна, кандидат технічних наук, завідуючий лабораторією GRID-технологій та хмарних обчислень Національного університету «Запорізька політехніка». Закінчила магістратуру за спеціальністю «Комп'ютерні системи та мережі». Захищено дисертацію за спеціальністю «Інформаційні технології» у 2019 році.
Автор 14 наукових публікацій Наукові інтереси – надійність інформаційних систем, телемедичні системи.
ORCID ID: 0000-0002-6499-6017

Olga PETROVA, PhD, Head of Laboratory of GRID-technologies and Cloud Computing of National University «Zaporizka politechnika». She has a master's degree in "Computer systems and networks". She finished PhD in "Information technology" in 2019. She is the author of 14 scientific publications. Scientific interests – reliability of informational systems, telemedical systems.
ORCID ID: 0000-0002-6499-6017

ШИТІКОВА Олена Вікторівна – кандидат технічних наук, ст. науковий співробітник Національного університету «Запорізька політехніка». Закінчила Запорізький національний технічний університет за спеціальністю «Програмне забезпечення автоматизованих систем».
Захищено дисертацію за спеціальністю «Інформаційні технології» у 2017 році. Автор 45 наукових праць. Основні напрямки наукових досліджень: інженерія програмного забезпечення, менеджмент ризиків складних технічних систем, автоматизація процесу випробувань складних технічних систем.
Olena SHYTIKOVA – Ph.D., Researcher of Software Tool Department at National University «Zaporizhzhia polytechnic». She graduated from Zaporizhzhya National Technical University with speciality Software Engineering. She finished PhD study in «Information technology», and defended thesis «Information technology of gas turbine unit for terrestrial usage test process with consideration of uncertainty influence» in 2017. She is the author of 45 scientific publications. Main research fields: System Engineering, Risk Management of Complex Technical Systems and Test Process, Automation of the test process

Galyna Tabunshchyk
Tetiana Kapliienko
Olga Petrova
Olena Shytikova

Design of Informational Infrusructure of Medical and Telemedical Systems

Students Textbook

Co-funded by the
Erasmus+ Programme
of the European Union

Навчальне видання

Табунщик Галина Володимирівна
Каплієнко Тетяна Ігорівна
Петрова Ольга Анатоліївна
Шитікова Олена Вікторівна

Проєктування інформаційної інфраструктури медичних та телемедичних систем

Навчальний посібник

Комп'ютерний набір Табунщик Г. В.

Co-funded by the
Erasmus+ Programme
of the European Union

Формат 60×84/16. Ум. друк. арк. 11,51.
Тираж 100 прим.

Видавець і виготівник ПП «Євро-Волинь»
м. Житомир вул. Крошенська, буд. 45, кв. 34
Свідоцтво серія ДК № 7208 від 07.12.2020