Національний університет «Запорізька політехніка»

Т.І. Каплієнко

АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лекційний матеріал

Запоріжжя, 2023

ЗМІСТ

ВСТУП

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 матеріалуу в тексті використовуються наступні позначення:

 – визначення

 – практичні завдання

 – приклади, контрольні запитання

 – рекомендована література

ЧАСТИНА 1. ОСНОВНІ ПОНЯТТЯ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1 ЖИТТЄВИЙ ЦИКЛ ТА РОЗРОБЛЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Етапи життєвого циклу програмного забезпечення

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 застосунків і серверів баз даних.

Коротко кажучи, аналіз вказує на те, що робити; проєктування – на те, як за допомогою наявної технології зробити це, а реалізація 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 застосунок 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 застосунку (*application model*)– опису видимих користувачеві частин самого застосунку. Наприклад, для застосунку 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 застосунку 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 застосунку, 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.

1.2 Підходи до розроблення програмного забезпечення

Революція в програмному забезпеченні викликала низку значних змін у способах роботи програмних продуктів. Зокрема, значно збільшилися інтерактивні можливості програм.

Розрізняють структурний і об'єктно-орієнтований підходи до розроблення ПЗ.

Структурний підхід до розроблення систем набув широкого поширення в 80х роках. Цей підхід заснований на двох методах: *діаграмах потоків даних* (*data flow diagrams*) для моделювання процесів і *діаграмах сутність-зв'язок* (*entity relationship diagrams*) для моделювання даних.

Структурний підхід є функціонально-орієнтованим і розглядає *DFD*-діаграми як рушійну силу розроблення ПЗ. У зв'язку з поширенням баз даних, значення *DFD*-діаграм у структурній розробці знизилося, і підхід став більш орієнтованим на дані, і, відповідно, акцент при розроблені ПЗ змістився на *ERD*-діаграми.

Поєднання *DFD* і *ERD* діаграм дає відносно повні моделі аналізу, що фіксують всі функції та дані системи на необхідному рівні абстракції незалежно від можливостей апаратного та програмного забезпечення. Потім модель аналізу перетвориться впроєктну модель, що зазвичай виражається в поняттях реляційних баз даних. Після цього слідує етап реалізації.

Недоліками цього підходу є:

– він є послідовним і трансформаційним, а не ітеративним підходом з нарощуванням можливостей;

– він направлений на постачання негнучких рішень;

– в ін передбачає розроблення «з чистого аркуша» і не підтримує повторного використання компонент.

Об'єктно-орієнтований підхід набув поширення в 1990-х роках. Асоціація виробників ПЗ *Object Management Group* (*OMG*) затвердила як стандартний засіб моделювання цього підходу мову *UML*.

Порівняно зі структурним підходом об'єктно-орієнтований підхід більшою мірою орієнтований на дані – він розвивається довкола моделей класів. Зростаюче значення використання в мові *UML* претендентів сприяє незначному зсуву акцентів від даних до функцій.

До найбільш важливих категорій застосунків, для яких потрібна об'єктна технологія, відносяться обчислювальна обробка для робочих груп і системи мультимедіа.

Об'єктний підхід до розроблення систем слідує ітеративному процесу з нарощуванням можливостей. Єдина модель конкретизується на етапах аналізу, проєктування та реалізації.

Об'єктно-орієнтований підхід призводить до виникнення низки труднощів:

– оскільки етап аналізу проводиться на ще вищому рівні абстракції і якщо серверна частина рішення з реалізації передбачає

використання реляційної бази даних, семантичний розрив між концепцією і її реалізацією може бути значним;

– керування проєктом складно здійснювати;

– висока складність рішення, що у свою чергу позначається на таких характеристиках ПЗ, як пристосованість до супроводу і масштабованість.

1.3 Об'єктно-орієнтована методологія та мова *UML*

Сутність об'єктно-орієнтованого підходу (ООП) полягає, перш за все, у двох моментах.

З одного боку, він надає розробнику інструмент, що дозволяє описати завдання й істотну частину реалізації проєкту в термінах, що характеризують предметну область, а не комп'ютерну модель.

З іншого боку, об'єктно-орієнтоване програмування надає розробникам гнучкий потужний універсальний інструмент, не пов'язаний з якимось певним класом задач.

1.3.1 Об'єктна модель

Як концептуальна основа об'єктно-орієнтованого аналізу та проєктування виступає об'єктна модель.

Об'єктну модель складають чотири головні компоненти:

– абстрагування;
– інкапсуляція;
– модульність;
– ієрархія.

Абстрагування дозволяє виділити суттєві характеристики деякого об'єкта, що відрізняють його від усіх інших видів об'єктів. Абстракція чітко визначає концептуальні кордони об'єкта з точки зору спостерігача.

Інкапсуляція – це процес відокремлення одне від одного елементів об'єкта, що визначають його будову.

Ієрархія – це впорядкування абстракцій, засіб класифікації об'єктів та систематизації зв'язків між об'єктами.

Модульність – це подання системи у вигляді сукупності відокремлених сегментів, зв'язок між якими забезпечується за

допомогою зв’язків між класами, що визначаються в цих сегментах.

Розглянемо всі ці компоненти більш докладно.

Абстрагування. Абстрагування є одним із основних методів, що використовуються для вирішення складних завдань. Абстракція виділяє істотні характеристики певного об’єкта, що відрізняють його від інших видів об’єктів і, таким чином, чітко визначає його концептуальні кордони з точки зору спостерігача.

Наприклад. Автомобіль має корпус і колеса, причому корпуси кожного автомобіля дуже відрізняються один від одного.

Під абстрактними моделями, що використовуються в ООП, розуміють моделі, які характеризуються найбільш загальними властивостями актуальними для практичних випадків.

ООП характеризується наявністю двох основних видів абстракцій:

– тип даних об’єктів природи (***клас***) – розширення вихідних типів мови програмування, що визначається програмістом;

– екземпляр класу (***об’єкт***) – змінна класу.

Об’єкт має стан, поведінку та ідентичність.

Стан об’єкта характеризується набором його властивостей (атрибутів) і поточними значеннями кожної з цих властивостей (автомобіль заведений, фари включено).

Стан об’єкта – результат його поведінки, тобто виконання впевній послідовності характерних для нього дій.

Ідентичність – це така властивість об’єкта, що відрізняє його від усіх інших об’єктів того ж типу.

Наприклад . Розглянемо абстракцію «Лампочка».

Для лампочки визначено, як мінімум, два стани: ввімкнуто та вимкнуто. Для того щоб з’ясувати її поточний стан, нам потрібно подивитись на лампочку. Процес вмикання та вимикання лампочки характеризує поведінку об’єкта. Всі лампочки мають приблизно однакову поведінку, хоча їх реалізації можуть відрізнятися.

На рівні абстрактної моделі кожній операції відповідає метод класу (рис. 2.1).

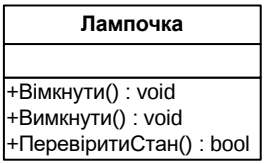

Рисунок 1.1 – Абстракція електричної лампочки

Клас визначає загальні властивості об’єктів (тобто тип). Коли покупець приходить до магазину та просить дати йому електричну лампочку, то висловлюючись термінами ООП, він оперує ім’ям класу (вказує тип предмета, що його цікавить). Продавець видає конкретну лампочку – об’єкт класу.

Отже, абстрагування відповідає за зовнішню поведінку об’єкта – так поводяться всі об’єкти класу.

Інкапсуляція. Інкапсуляція тісно пов’язана з абстракцією. Якщо абстрагування спрямоване на спостереження поведінки об’єкта, то інкапсуляція займається внутрішнім пристроєм і тому визначає чіткі межі між різними абстракціями.

Інша важлива властивість інкапсуляції – можливість приховування реалізації. Інкапсуляція завжди припускає можливість обмеження доступу до даних класу. З одного боку, це дозволяє спростити інтерфейс класу, показавши найбільш істотні для зовнішнього користувача дані та методи. З іншого боку, приховування реалізації забезпечує можливість внесення змін уреалізацію класу без зміни інших класів.

Наприклад. Для класу «Лампочка» рівень реалізації зв’язаний зі способом уявлення стану лампочки. У нашому прикладі ми ввели логічну змінну. Знак «мінус» у атрибута *стан* показує, що ззовні доступ до нього закритий (рис. 2.2). Це значить, що змінити значення змінної *стан* можуть тільки методи класу.

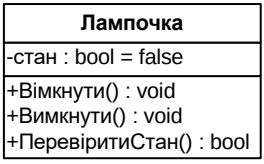

Рисунок 1.2 – Клас «Лампочка»

Отже робимо висновок, що інкапсуляція ізолює інтерфейс від реалізації, вона зв'язана з керуванням доступом до даних класу. Але важливо мати на увазі, що інкапсуляція не забезпечує повної закритості класу.

Види структурних ієрархій. Відомо, що вивчення і організація складних технічних систем істотно спрощується, якщо слідувати принципам функціонально-ієрархічної декомпозиції.

Ієрархічна декомпозиція й ієрархічна організація ПЗ утворюють один із основних систематичних методів подолання складності останнього.

Структурна ієрархія «*частина-ціле*». *С*труктурна ієрархія побудована за принципом «*частина-ціле*» називається ***агрегацією***.

Агрегація є найпростішою реалізацією однієї з базових ідей, що лежать в основі ООП: створений клас в ідеалі повинен являти собою фрагмент корисного коду, доступного для повторного використання.

Варіант агрегації, при якому відношення володіння чітко виражене, а час життя частин і цілого збігаються, називається композицією. Проста агрегація на відміну від композитної агрегації припускає, що одна і та ж «частина» може одночасно належати різним «цілим».

Відношення композиції утворюють, наприклад, мобільний діагностичний комплекс як ціле, а монітор функцій пацієнта, безконтактний інфрачервоний термометр, транспортна валіза як його частини (рис. 2.3).

Рисунок 1.3 – Приклад композиції

Механізм агрегування надає програмістові гнучкий засіб організації ієрархії класів.

Структурні ієрархії «is-a» і «is-like-a». Ідея спадкування випливає з бажання мати зручний засіб, що б дозволив створювати клас, дані і поведінка якого в деякій частині схожі з існуючим класом.

Спадкування є найпоширенішою в об'єктно-орієнтованих програмах формою відносини узагальнення. Узагальнення означає, що об'єкти класу нащадка можуть використовуватися скрізь, де допустимі об'єкти батьківського класу. Зворотне в загальному випадку неправильно. Наприклад, коли нам потрібно створити клас «Кольорова лампочка», що, крім стану, має ще колір, не варто створювати новий клас. Треба створити клас-нащадок від класу «Лампочка», якій містить специфічний атрибут колір та специфічні методи (рис. 2.4).

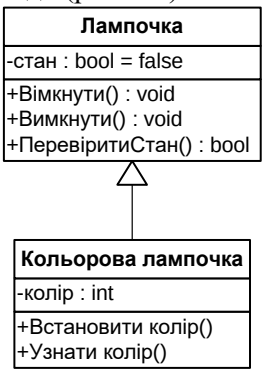

Рисунок 1.4 – Приклад спадкування

Об'єкт класу «Кольорова лампочка» успадкує всі дані та методи батьківського класу (рис.2.5).

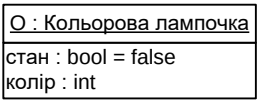

Рисунок 1.5 – Об'єкт класу «Кольорова лампочка»

Створюючи базовий тип (базовий клас), програміст висловлює найбільш загальні ідеї щодо об’єктів, з яких конструюється програма. У похідних класах програміст уточнює відмінність уреалізації конструкцій базового класу.

При конструюванні нового класу програміст висловлює найбільш загальні ідеї щодо об’єктів, з яких від базового створюється новий тип. Цей тип включає всі дані і методи базового типу. Відповідність типу похідного класу типу базового класу є одним із фундаментальних принципів реалізації концепції успадкування в рамках об’єктної моделі.

Поведінку об’єктів, для яких викликаються видозмінені методи, інтерфейс, до яких заявлено в базовому класі, називають поліморфною, а механізм, що забезпечує таку поведінку, називають поліморфізмом.

Ієрархія і модулі. Під модульністю будемо розуміти уявлення системи у вигляді сукупності відокремлених сегментів, зв’язок між якими забезпечується за допомогою зв’язків між класами, що визначаються в цих сегментах. При цьому виділяють два рівні модульності: фізичний та логічний.

Фізичний рівень має справу зі структурою файлів і каталогів, уяких зберігаються окремі модулі, що становлять проєкт.

Механізм забезпечення реалізації модульності програмних проєктів на основі логічної реалізації програмних об’єктів зазвичай пов’язаний з використанням простору імен.

Простір імен – це іменована або неіменована відкрита область дії імен.

Побудова модуля завжди припускає процес логічного групування, оголошень та визначень у зв’язку з розв’язуваними завданнями йспільністю використовуваних даних.

Простору імен надають механізм відображення логічного групування програмних об’єктів засобами мови програмування.

1.3.2 Історія об’єктно-орієнтованого моделювання та мови UML

Мови для об’єктно-орієнтованого моделювання почали з’являтися в період із середини 1970 року й кінця 1980-х років,

коли різні розроблювачі експериментували з різними підходами до об'єктно-орієнтованого аналізу й проєктування.

В основі *UML* лежить кілька об'єктно-орієнтованих методів, кожний з яких спочатку був орієнтований на підтримку окремих етапів об'єктно-орієнтованого аналізу й проєктування (ООАП):

– метод Граді Буча (Grady Booch), умовна назва *Booch* (*Booch'91, Booch Lite, Booch'93*), вважався найбільш ефективним на етапах проєктування й розроблення програмних систем;

– метод Джеймса Румбаха (*James Rumbaugh*), *Object Modeling Technique* (*OMT*, пізніше *OMT-2*), оптимально підходив для аналізу процесів оброблення даних у інформаційних системах;

– метод Айвара Джекобсона (Ivar Jacobson), *Object-Oriented Software Engineering* (*OOSE*), містив засоби подання прецедентів, що мають істотне значення на етапі аналізу вимог при проєктуванні бізнесів- застосунків.

Історія розвитку *UML* датується 1994 р., коли почалася інтеграція/уніфікація вищевказаних методів. Проєкт уніфікованого методу (*Unified Method*) версії 0.8 був опублікований ужовтні 1995 р.

Всі питання розроблення й супроводу мови *UML* сконцентровані в рамках консорціуму *OMG (Object Management Group*). Хоча *OMG* був створений з метою розроблення пропозицій щодо стандартизації об'єктних і компонентних технологій *CORBA*, мова *UML* набула статусу другого стратегічного напрямку в роботі консорціуму. Улистопаді 1997 р. *OMG* оголосив *UML* стандартною мовою об'єктно-орієнтованого моделювання та взяв на себе обов'язок з його подальшого розвитку. Група фахівців забезпечує публікацію описів наступних версій мови *UML* і запитів пропозицій *RFP* (*Request For Proposals*) по його стандартизації. Статус мови *UML* визначений як відкритий для всіх пропозицій щодо доопрацювання йудосконалення. У 2003р. як результат розгляду набору запитів *RFP* 2000р. був опублікований опис мови *UML* 2.0, що містить інфраструктуру *UML*, мова обмежень об'єктів (*Object Constraint Language – OCL*), суперструктуру *UML* і формат обміну даними.

Основними ініціативами консорціуму *OMG* у рамках роботи над проєктом *UML* є:

1) моделювання систем реального часу;

2) визначення моделі виконання – точної специфікації поведінки моделей, що підтримуються *UML*;

3) оброблення даних підприємства – визначені так звані профілі, що описують способи створення великих розподілених паралельних систем підприємства;

4) визначення процесу розроблення програмного забезпечення – специфіковані структури визначення процесів розробки програмного забезпечення;

5) стандарт зберігання даних;

6) зіставлення технології *CORBA* та мови *UML*;

7) формат *XMI (Metadata Interchange*) для обміну моделями *UML* у текстовому форматі [9].

Існує консорціум партнерів *UML* (D*igital Equipment Corp., HP, Intellicorp, IBM, ICON Computing, Microsoft, Oracle, Rational Software і*інші), що забезпечують уточнення нотації, удосконалення й доповнення мови, а також супровід розробки інструментальних засобів підтримки. Особливе місце займає компанія *Rational Software Corporation*, що реалізувала *Rational Rose 98* – один з перших інструментальних CASE-засобів, уякому була підтримана мова *UML*. Необхідно відзначити увагу компанії Microsoft до технології, що базується на мові *UML*, на основі якої створена інформаційна модель (*UML Information Model*), призначена для створення стандартного інтерфейсу між засобами розроблення застосунків і засобами візуального моделювання. На даний момент *Rational Software Corporation* офіційно входить до складу *IBM* (http://www-306.ibm.com/software/rational/).

Мова *UML* призначена для вирішення таких завдань:

1)надати в розпорядження користувачів готову до використання виразну потужну мову візуального моделювання, що дозволяє розробляти осмислені моделі й обмінюватися ними;

2)передбачити внутрішні механізми розширення й спеціалізації базових концепцій мови;

3)забезпечити максимальну незалежність проєкту створення програмного забезпечення від конкретних мов програмування йпроцесів розробки;

5)забезпечити формальну основу для однозначної інтерпретації мови;

6)стимулювати розширення ринку об'єктно-орієнтованих інструментальних засобів створення програмного забезпечення;

7)інтегрувати кращий практичний досвід використання мови йреалізації програмних засобів його підтримки.

Принципи використання *UML* специфіковані в *Rational Unified Process* (*RUP*) – розвинутій методиці створення програмного забезпечення, оформленої у вигляді бази знань, фізично розміщеної на web (http://www-306.ibm.com/software/awdtools/rup/support/) іоснащеною пошуковою системою.

Основним документом з *UML* є [3], де описана метамодель *UML* ідуже мало уваги приділяється семантиці мовних конструкцій. Опис поточної версії мови *UML* і приклади розроблення програмних систем з використанням *CASE*-засобу *Rational Rose* можна знайти на web-вузлі *OMG* (www.omg.org); модифікації мови *UML* і його новітні версії – на вузлі www.celigent.com/uml.

UML створювалася як мова моделювання загального призначення для застосування в таких «дискретних» галузях, як програмне забезпечення, апаратні засоби й цифрова логіка. Структури *UML* дозволяють фіксувати різноманітні рішення звідображення:

1) функціональності системи;

2) динамічної та статичної структури системи;

3) організації елементів системи;

4) реалізації системи.

Популярності набуває використання *UML* при проєктуванні баз даних. Завдяки відкритості (наявності в мові механізмів розширення) він надає потужний інструментарій для вирішення завдань інших галузей, наприклад, бізнесу-моделювання.

1.4 Agile розробка інформаційних систем

Ітеративна розробка – це технічний підхід до створення програмних систем, покладений в основу опису об'єктно-орієнтованого аналізу та проєктування.

У рамках цього підходу розроблення виконується у вигляді декількох короткострокових міні-проєктів фіксованої тривалості, що називаються *ітераціями*. Кожна ітерація містить свої власні етапи аналізу вимог, проєктування, реалізації та завершується тестуванням, інтеграцією та створенням робочої системи.

Перші ідеї ітеративного процесу називалися «проєктування по спіралі й еволюційною розробкою».

Переваги ітеративного розроблення:

– своєчасне усвідомлення можливих технічних ризиків, осмислення вимог, завдань проєкту та зручності використання системи;

– швидкий та помітний прогрес;

– ранній зворотний зв'язок, можливість обліку побажань користувачів і адаптації системи. У результаті система більш задовольняє реальні вимоги керівників і споживачів;

– керована складність;

– отриманий при реалізації кожної ітерації досвід можна методично використовувати для поліпшення самого процесу розроблення.

Agile – гнучка методологія розроблення, – це концептуальний каркас, в рамках якого виконується розробка програмного забезпечення. Основна ідея всіх гнучких моделей полягає в тому, що процес, який застосовується у розробці ПЗ, повинен бути адаптивним. Вони декларують своєю вищою цінністю орієнтованість на людей та їх взаємодію, а не на процеси і засоби.

1.4.1 Модель керування програмними проєктами SCRUM

Методологія *Scrum* була вперше озвучена в роботі Хіротака Такеучи й Ікуджіро Нонаки, опублікованій в *Harvard Business Review*. Джеф Сазерленд використовував цю роботу при створенні методології для корпорації *Easel* у 1993 році, яку за аналогією і назвав *Scrum*, а Кен Швабер формалізував процес для використання в індустрії розроблення програмного забезпечення.

Мета цієї методології – виявити й усунути відхилення від бажаного результату на більш ранніх етапах розроблення програмного продукту.

Методологія *Scrum* визначає:

1. Правила, за якими повинен плануватися і управлятися список вимог до продукту, з метою досягнення максимальної прибутковості від реалізованої функціональності;
2. Правила планування ітерацій для забезпечення максимальної зацікавленості команди в отриманні результату;
3. Основні правила взаємодії учасників команди для максимально швидкої реакції на непередбачені робочі ситуації;
4. Правила аналізу і коригування процесу розроблення для вдосконалення взаємодії всередині команди.

Кожну ітерацію можна описати так: «Плануємо – Фіксуємо – Реалізуємо – Аналізуємо». За рахунок фіксування вимог на протязі однієї ітерації та зміни довжини ітерації можна керувати балансом між гнучкістю та плануємістю розроблення.

Scrum фокусується на постійному визначенні пріоритетних завдань, ґрунтуючись на бізнес цілях, що збільшує корисність і прибутковість проєкту на його ранніх стадіях. Зважаючи на те, що при ініціації проєкту його прибутковість визначити майже неможливо, *Scrum* пропонує концентруватися на якості розроблення і до кінця кожної ітерації мати проміжний продукт, який можна використовувати. Наприклад, результатом ітерації може бути каркас сайту, який можна показати на презентації.

Методологія *Scrum* орієнтована на те, щоб оперативно пристосовуватися до змін у вимогах, що дозволяє команді швидко адаптувати продукт до потреб замовника.

Девіз *Scrum* – «аналізуй та адаптуй»: аналізуйте те, що отримали, адаптуйте те, що вже розроблено до реальних потреб, а потім аналізуйте знову. Чим менше формалізму, тим більш гнучко та ефективно можна працювати, – це основний принцип даної методології. Але це не означає, що формальних процесів не повинно бути зовсім, їх має бути достатньо для організації ефективної взаємодії і керування проєктом. Формальна частина *Scrum* складається з трьох ролей, трьох практик і трьох основних документів.

В *Scrum*-проєктах відділяють наступні ролі:

1. **Власник продукту** (*Product Owner*) – людина, що формулює вимоги програмістам. Зазвичай власник продукту є

представником або довіреною особою замовника, а для компаній, що випускають коробкові продукти, він являє собою ринок, на якому реалізується продукт. Власник продукту повинен скласти бізнес план, який показує очікувану дохідність і план розвитку до вимог, відсортованими за коефіцієнтом окупності інвестицій. Виходячи з наявної інформації, власник продукту готує перелік вимог, відсортований за значимістю;

2. *Scrum*-**майстер** (*Scrum Master*) – забезпечує максимальну працездатність і продуктивність команди, чітку взаємодію між усіма учасниками проєкту, своєчасне вирішення всіх проблем, що гальмують або зупиняють роботу будь-якого члена команди, відгороджує ко проходження процесу всіх учасників проєкту. Ця людина має бути одним з членів команди розроблення і брати участь у проєкті як розробник. Він відповідає за своєчасне вирішення поточних проблем.
3. *Scrum*-**команда** (*Scrum Team*) – група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Перше завдання цієї команди – поставити реально досяжну, прогнозовану, цікаву і значущу мету для ітерації. Друге завдання – зробити все для того, щоб ця мета була досягнута у відведені терміни і з заявленою якістю. Мета ітерації вважається досягнутою тільки в тому випадку, якщо всі поставлені завдання реалізовані, весь код написаний за певними проєктом «стандартам кодування» (*coding guidelines*), програма протестована повністю, а всі знайдені дефекти усунені. Програмісти цієї команди повинні вміти оцінювати та планувати свою роботу, працювати в команді, постійно аналізувати та поліпшувати якість взаємодії та роботи. В обов'язки всіх членів *Scrum*-команди входить участь у виборі мети ітерації і визначення результату роботи. Вони повинні робити все можливе і неможливе для досягнення мети ітерації в рамках, визначених проєктом, ефективно взаємодіяти з усіма учасниками команди, самостійно організовувати свою роботу, надавати власнику робочий продукт в кінці кожного циклу.

Scrum містить наступні документи: **журнал продукту** (*Product Backlog*), **журнал спринту** (*Sprint Backlog*) та **графік спринту** (*Burndown Chart*).

На початку проєкту власник продукту готує **журнал продукту** (табл. 1.1) – перелік вимог, відсортований за значимістю, а *Scrum*-команда доповнює цей журнал оцінками вартості реалізації вимог. Перелік повинен містити функціональні та технічні вимоги, необхідні для реалізації продукту. Найбільш пріоритетні з них повинні бути досить детально прописані, щоб їх можна було оцінити і протестувати. Про своєчасну деталізацію вимог має дбати власник продукту і надавати необхідний обсяг у потрібний час. У цьому сенсі програмісти є замовниками вимог для власника продукту. Надалі інші вимоги повинні поступово уточнюватися та деталізуватися до такого ж рівня. Головне, щоб у команди завжди був достатній обсяг підготовлених до реалізації вимог.

Таблиця 1.1. Приклад журналу продукту

Номер вимоги	**Опис вимоги**	**Цінність бізнесу (ум. од.)**	**Пріоритет**	**Високорівнева оцінка (часи)**
*FE*1	Покупець може зареєструватися на сайті	10.000	1	20
*FE*2	Покупець може ввести свої персональні дані	12.000	6	36
*FE*3	Покупець може побачити список доступних виробів	16.000	10	30
*FE*4	Покупець може купити виріб	100.000	20	48
*FE*5	Покупець може робити пошук виробів	80.000	30	32
*FE*6	Покупець може підписатись на новини	30.000	40	66

Графік спринту дозволяє також власнику продукту спостерігати за ходом ітерації – якщо загальний обсяг робіт не зменшується щодня, значить, щось йде не так. Під час сесії планування команда знаходить і оцінює завдання, які треба виконати для успішного завершення ітерації. Сума оцінок всіх завдань в журналі спринту є загальним обсягом роботи, який треба виконати за ітерацію. Піс ля завершення кожного завдання *Scrum*-майстер перераховує обсяг роботи, що залишилася, і зазначає це на графіку спринту. Тільки в тому випадку, якщо по закінченні ітерації у журналі спринту не залишилося незавершених завдань, ітерація вважається успішною. Графік спринту використовується як допоміжний інструмент, що дозволяє коригувати роботу для завершення ітерації вчасно, з працюючим кодом і необхідною якістю.

К *Scrum*-**практикам** відносяться: спринт (*Sprint*), скрам (*Daily Scrum Meeting*) та демонстраційне засідання (*Sprint Review Meeting*).

1. Підготовка до першої ітерації, званої **спринт** (*Sprint*), починається після того, як власник продукту розробив **журнал продукту**.

При плануванні ітерації відбувається детальне розроблення сесій планування спринту (*Sprint Planning Meeting*), яке починається з того, що власник продукту, *Scrum*-команда і *Scrum*-майстер перевіряють план розвитку продукту, план релізу та перелік вимог, *Scrum*-команда перевіряє оцінки вимог, переконується, що вони досить точні, щоб почати працювати, вирішує, який обсяг роботи вона може успішно виконати за спринт, ґрунтуючись на розмірі команди, доступному часі та продуктивності. *Scrum*-команда повинна вибирати перші за пріоритетом вимоги з журналу продукту. Після того як *Scrum*-команда зобов'язується реалізувати обрані вимоги, *Scrum*-майстер починає планування спринту. *Scrum*-команда розбиває вибрані вимоги на завдання, необхідні для його реалізації. Ця активність в ідеалі не повинна займати більше чотирьох годин, і її результатом є **журнал спринту**. Необхідно, щоб всі учасники команди взяли на себе зобов'язання з реалізації обраної мети.

2. Після закінчення планування починається ітерація. Кожен день *Scrum*-майстер проводить «**скрам**» (*Daily Scrum Meeting*) – п'ятнадцятихвилинної нарада, мета якої – досягти розуміння того, що сталося з часу попередньої наради, скоригувати робочий план до реалій сьогодення і позначити шляхи вирішення існуючих проблем. Кожен учасник *Scrum*-команди відповідає на три питання: що я зробив з часу попереднього скрама, що мене гальмує або зупиняє в роботі, що я буду робити до наступного скрама? У цій нараді може брати участь будь-яка зацікавлена особа, але тільки учасники *Scrum*-команди мають право приймати рішення. Правило обґрунтовано тим, що вони давали зобов'язання

випадку заощаджується час на перемикання команди з активного розроблення на планування та демонстраційні мітинги. Якщо вимоги часто змінюються і доповнюються, потрібно відштовхуватися від двотижневого циклу, в будь-якому випадку довжина ітерації – це величина експериментальна.

Недоліки *Scrum*:

1. Складно домогтися активної участі від кожного розробника і злагодженої колективної роботи в команді;
2. Складно залучити постачальника вимог до активної участі в проєкті, зацікавити його динамікою розвитку продукту, дати можливість бути активним вболівальником і спонсором команди.

Проте, незважаючи на це, використання методології *Scrum* в проєктах дозволяє:

—використовувати весь технічний багаж, накопичений компанією, тому що головний напрям методології *Scrum* направлений на керування проєктами і не задає ніяких технічних практик;

—з високою якістю та в рамках бюджету реалізувати великий обсяг функціональності та специфікації, які були відсутні на момент початку проєкту;

—забезпечити максимальну бізнес-цінність виробленого продукту, за рахунок того, що практично всі реалізовані функції активно використовуються відвідувачами;

— досягнути високу супровідність коду (можливість внесення змін з мінімальними трудовитратами) – вартість змін, внесених

до продукту, практично еквівалентна вартості розроблення аналогічних функцій продукту на початку проєкту, що рідко буває у RUP чи MSF моделях виробництва, для яких характерний експонентний ріст вартості змін по мірі виконання проєкту

1.4.2 Модель керування програмними проєктами Kanban

З початку 70-х років у Японії, а потім і в інших країнах набула великого поширення система «*Канбан*», що є механізмом організації безперервного гнучкого виробничого потоку і функціонує практично без страхових запасів. Традиційна концепція організації виробництва, як відомо, спрямована на запобігання простоям та організацію безперервного потоку з обов'язковим створенням страхового запасу. Японська концепція ґрунтується на практично повній відмові від страхових запасів. Більше того, менеджери навмисне дають робітникам змогу повністю випробувати на собі наслідки простоїв. В результаті весь персонал постійно зайнятий виявленням причин збоїв у виробництві та пошуком шляхів підвищення надійності та запасу міцності системи управління. Після виявлення та усунення причин простоїв керівники ще більше скорочують страховий запас, породжуючи додаткові зусилля з покращення організації виробництва з боку всього персоналу. В умовах системи «*Канбан*», на відміну від традиційного підходу, виробник не має завершеного плану й графіка виробництва, а жорстко пов'язаний конкретним замовленням споживача. Він не взагалі оптимізує свою роботу, а в межах замовлення. Конкретного графіка роботи на декаду, місяць він не має. Кожен зайнятий у технологічному ланцюгу робітник знає, що він вироблятиме продукцію тільки тоді, коли карта «*Канбан*» з його продукції відкріплена від контейнера на складі, тобто коли продукція фактично надійшла на наступну стадію обробки.

Конкретний графік послідовності праці одержують лінії кінцевого складання, вони розкручують клубок інформації у зворотному напрямі. Іншими словами, графіки виробництва не переглядаються, а тільки формуються рухом карток «*Канбан*»,

тому що зняття карти відбору продукції, графіка виготовлення її фактично не було. Виробництво постійно перебуває в стані надбудови, відбувається його системне настроювання під зміни ринкової кон'юнктури.

Методика розробки програмного забезпечення ***Канбан*** була введена у практичне використання Девідом Андерсеном у 2007 році. Багато з цих практик та підходів використовувалися різними *Agile* командами, перш ніж були описані як єдине ціле.

Нововведення полягали в тому, що було введено завдання «в процесі». Це робилося і раніше іншими *Agile*-командами, але в Канбан існує всім відоме обмеження на кількість робочих завдань, які можуть виконуватися в один час. Ця межа зазвичай досить низька – ліміт приблизно дорівнює числу розробників в команді або трохи менший.

Основні положення Канбан наступні:

1. Візуалізація потоку робіт:

В кінці кожного спринту проводиться демонстраційне засідання (Sprint Review Meeting) тривалістю не більше чотирьох годин. Спочатку Scrum-команда демонструє власнику продукту зроблену протягом спринту роботу, а той у свою чергу веде цю частину наради і може запросити до участі всіх зацікавлених осіб. Власник продукту визначає, які вимоги з журналу спринту були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в журналі продукту для наступної ітерації. У другій частині мітингу проводиться аналіз минулого спринту, який веде Scrum-майстер. Результатом є:

– розбиття роботи на частини, кожна частина виписується на картку і прикріплюється до стіни;

– розбиття усіх завдань на стовпчики для розділення по стадіям розробки.

2. Обмеження незавершеної роботи (НЗР) (work-in-progress) визначається можлива кількість незавершених пунктів на кожній стадії робочого процесу.

3. Вимірювання часу виконання завдання (lead time) (середньої тривалості часу для завершення одного пункту, іноді звану «оперативним часом» (cycle time)), оптимізація процесу, щоб звести час виконання завдання до мінімуму і зробити його настільки прогнозованим, наскільки це можливо.

Канбан – це не конкретний процес, а система цінностей. Його можна описати однією простою фразою – «Зменшення виконується в певний момент роботи (work in progress)».

Різниця між Канбан і Scrum:

— Канбан немає таймбоксов (ні на завдання, ні на спринти);

— Канбан завдання більше і їх менше;

— Канбан оцінки термінів на задачу опціональні або взагалі їх немає;

— Канбан «швидкість роботи команди» відсутня і враховується тільки середній час на повну реалізацію завдання.

Команда для роботи використовує Канбан-дошку. Наприклад, вона може виглядати рис.

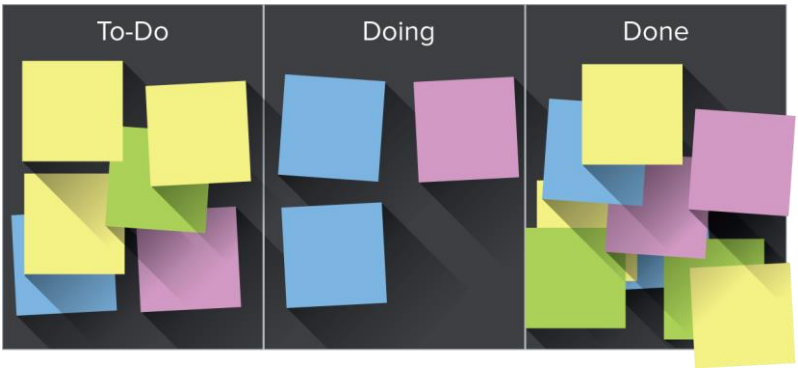

Рисунок 1.6. – Приклад Канбан дошки

Цілі проєкту. Сюди можна помістити високорівневі цілі проєкту.

Черга завдань. Тут зберігаються завдання, які готові до того, щоб почати їх виконувати. Завжди для виконання береться зверху, сама пріоритетна задача і її картка переміщується в наступний стовпець.

Опрацювання дизайну. Цей та інші стовпці до «Закінчено» можуть змінюватися, тому що саме команда вирішує, які кроки проходить завдання до стану «Закінчено». Наприклад, в цьому стовпці можуть перебувати завдання, для яких дизайн коду або

інтерфейсу ще не ясний і обговорюється. Коли обговорення закінчені, завдання пересувається в наступний стовпець.

Розробка. Тут завдання перебуває до тих пір, поки розробка не завершена. Після завершення вона пересувається в наступний стов і і негативні способи спільної роботи, аналізує їх, робить висновки і приймає важливі для подальшої роботи рішення. Scrum-команда також визначає програми, які можуть працювати краще, і шукає шляхи для збільшення ефективності подальшої роботи. Потім цикл замикається, і починається планування наступного спринту.

Час між ітераціями – це час прийняття основоположних рішень, що впливають на хід всього проєкту. Під час спринту ніякі зміни ззовні не можуть бути зроблені. Після того як команда дала зобов'язання реалізувати журнал спринту, він фіксується, і зміни в ньому можуть бути зроблені тільки з таких причин:

– Scrum-команда протягом ітерації отримала кращу уяву про вимоги і потребує додаткових завдань для успішного завершення ітерації;

– знайдені дефекти, які треба обов'язково виправити для успішного завершення ітерації;

– Scrum-майстер і Scrum-команда можуть вирішити, що невеликі зміни, які не впливають на загальний обсяг робіт, можуть бути реалізовані в зв'язку з виниклою у власника продукту необхідністю.

Виходячи з того що журнал спринту не може бути змінений ззовні під час ітерації, потрібно вибирати його довжину, ґрунтуючись на стабільності вимог. Якщо вимоги стабільні, змінюються або доповнюються рідко, то можна вибрати шеститижневий цикл. У цьому стовпець і програмісти, які закінчили нову задачу, вже не зможуть перемістити її в стовпець тестування, тому що він заповнений. Для вирішення цієї проблеми, наприклад, програмісти можуть допомогти тестерам завершити одне із завдань тестування і тільки тоді пересунути нове завдання на звільнене місце. Це дозволить виконати обидва завдання швидше.

Можна обчислити час на виконання усередненої задачі. Необхідно позначати на картці дату, коли вона потрапила в чергу завдань, потім дату, коли її взяли в роботу і дату, коли її завершили. За цими трьома точками для хоча б 10 завдань можна

порахувати середній час очікування в черзі завдань і середній час виконання завдання.

Підсумовуючи, переваги *Канбан* полягають у тому, що:

1. Легко визначаються проблемні області.

2. Зростає продуктивність роботи за рахунок того, що члени команди техпідтримки самостійно організують свою роботу, використовуючи канбан-дошку, а менеджер лише спрямовує зусилля на приорітезацію великих проєктів і вирішення виникаючих проблем.

Недоліки *Канбан*:

1. Зі зменшенням НЗР з'являються обмеження – для менш пріоритетних проєктів не вистачає ресурсів, що призводить до скорочення кількості проєктів на команду.

2. Канбан накладає менше обмежень, ніж *Scrum*, тобто керівництво отримує більше параметрів для налаштування. Це може бути як недоліком, так і перевагою, залежно від ситуації.

3. Канбан – це ще більш «гнучка» методологія, ніж *Scrum* і *XP*, тому вона не підійде командам та проєктам, не готовим до гнучкої роботи.

1.5 Практичні завдання

Завдання 1. Напишіть реферат на одну із тем:

– призначення та використання діаграм потоків даних;

– проєктування та аналіз систем з використанням IDEF технологій;

– моделювання бізнес-процесів;

– методології структурного аналізу;

– функціонально-орієнтований аналіз.

Завдання 2. Зробіть порівняльну характеристику структурного та об'єктно-орієнтованого підходів; функціонально-орієнтованого та об'єктно-орієнтованого.

Задавдання 5. Дослідить характеритики підходу ScrumBan. Порівняйте його з підходами Scrum та Канбан.

1.6 Контрольні запитання

1.Які етапи містить життєвий цикл програмного забезпечення?

2. На якому етапі життєвого циклу здійснюється розроблення та інтеграція функціональних моделей та моделей даних для системи?

3. Який етап містить написання коду клієнтських програм?

4. На яких методах заснований структурний підхід?

5. Які недоліки структурного підходу?

6. На що орієнтований об'єктно-орієнтований підхід?

7. Яка мова затверджена як стандарт для об'єктно-орієнтованого підходу

8. З чого складається аналітична модель?

9. В чому полягає мета проєктування?

10. Порівняйте Scrum та Канбан.

1.7 Література до розділу

1. Larman C. Applying UML Patterns : An Introduction to Object-Oriented Analysis, Design and Iterative Development. - Delhi: Pearson India, 2005. - 796 p.

2. Maciaszek L. Requirements Analysis and System Design:Developing Information Systemswith UML with Using UML:Software Engineering with Objects and Components / L. Maciaszek, P. Stevens, Dr R. Pooley. – "Addison Wesley", 2003. – 432 p.

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

4. Табунщик, Г.В. Інженерія якості програмного забезпечення:навчальний посібник / Г.В. Табунщик, Р.К. Кудерметов, Т.І. Брагіна. – Запоріжжя: ЗНТУ, 2013. – 176 с.

5. Грицюк Ю. І. Аналіз вимог до програмного забезпечення. Навчальний посібник / Ю.І. Грицюк // Львів: Видавництво Львівської політехніки, 2018. - 456 с.

6. Определение требований с IBM Rational Requirements Composer: [Електрон. ресурс]. - Режим доступа: http://www.interface.ru/home.asp?artId=23530

7. Rational Requirements Composer [Електрон. ресурс]. - Режим доступа: https://jazz.net/products/rational-requirements-composer

8. What's New in Rational Requirements Composer Overview 2.0: [Електрон. ресурс]. - Режим доступа: https://www.youtube.com/watch?v=t2De-NHNVrs

9. Murray D. 30 productivity tips from the developers of IBM Rational Requirements Composer, Part 1. Navigation [Електрон. ресурс] / D. Murray, C. Geiss // Режим доступа: https://www.ibm.com/developerworks/rational/library/09/requirementscomposer30tipsnavigation/index.html?mhq=30%20productivity%20tips

10. Kruk N. 30 productivity tips from the developers of IBM Rational Requirements Composer, 30 productivity tips from the developers of IBM Rational Requirements Composer Part 3. Templates [Електрон. ресурс] / N. Kruk, D. Murray // Режим доступу: https://www.ibm.com/developerworks/rational/library/09/requirementscomposer30tipstemplates/index.html

11. Опис курсу: Основи IBM Rational Requirements Composer: [Електрон. ресурс]. - Режим доступу: https://www.ibm.com/developerworks/ru/rational/newto/.

12. Застосування розкадровок (Storyboard) в інструменті IBM Rational Requirements Composer [Електрон. ресурс]. - Режим доступу: http://www.ibm.com/developerworks/ru/library/r-l118_zhuo/index.html

13. DOORS Rational Requirements Composer > Tutorials [Електрон. ресурс]. - Режим доступу: https://www.almtoolbox.com/requirements-composer-tutorials.php

14. Соммервилл И. Инженерия программного обеспечения ; пер. с англ. [Текст] / СоммервиллИ. – М. : Изд. дом «Вильяме», 2002. – 624 с.

15. Bahrami, A. Object Oriented Systems Developoment? Irwin McGrawHill, 1999. – 412 pp.

16. Object Management Group: [Електрон. ресурс]. – Режим доступу:http://www.omg.org/.

17. Брагина, Т.И. Сравнительный анализ итеративных моделей разработки программного обеспечения [Текст] / Т.И. Брагина, Г.В. Табунщик // Радиоелектроніка. Інформатика. Управління. – 2010. – № 2. – С. 130 – 139.

18. Брагина, Т.И. Анализ подходов к управлению рисками в программных проєктах с итеративным жизненным циклом [Текст] / Т.И. Брагина, Г.В. Табунщик // Радиоелектроніка. Інформатика. Управління. – 2011. – №2 – С. 120-124.

19. Bragina, T. A Modified Method for Estimating Software Projects Labor Costs [Текст] / T. Bragina, G.Tabunshchik // TCSET'2012: Proc. Of XI Int. Conf. Modern Problems of Radio Engineering, Telecommunications and Computer Science. – Lviv: Lviv Polytechnic National University, 2012. – P. 245.

20. Брагіна, Т.І. Розробка засобів інтеграції / Т.І. Брагіна // Системи обробки інформації. Вип.8 (106). – 2012. – С. 127-130.

21. Брагина, Т.И. Оценка прогресса разработки программных проєктов в JIRA / Т.И. Брагина, К.В. Задорожная // Тиждень науки – 2013: зб. тез доп. щоріч. наук.-практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ (Запоріжжя, 20–25 квіт. 2013 р.). – Запоріжжя: ЗНТУ, 2013. – С. 270-271.

ЧАСТИНА 2. МЕТОДОЛОГІЧНІ ОСНОВИ АНАЛІЗУ, ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

2 КОНЦЕПТУАЛІЗАЦІЯ СИСТЕМИ

2.1 Розроблення концепції системи

Концептуалізація системи – це зародження застосунку.

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 застосунок – визначаються зацікавлені особи, які є 2-х типів: спонсори та користувачі.

– яке завдання буде вирішувати застосунок – 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 застосунку 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 застосунок?

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

2. Яке завдання буде вирішувати застосунок?

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 застосунків (JAD-метод);

– швидке розроблення програмних застосунків (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 3.1 – 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 застосунків (*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 застосунків (*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 3.2 Матриця залежності вимог

Вимоги	В1	2	3	4
В1	X	X	X	X
2	Конфлікт	X	X	X
3			X	X
4		Перекриття	Перекриття	X

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

Перекриваються вимоги 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- застосунків, 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-магазину іграшок. Дайте відповіді на такі запитання:

Для кого призначений цей застосунок? 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 «Інтернет-магазин подарунків» сформулюйте типові види ризиків.

Завдання 5. Виконайте завдання 1 для системи віртуального казино.

Завдання 6. Виконайте завдання 1 для системи дистанційного навчання.

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

Завдання 8. Для системи «*Диспетчер таксі*» сформулюйте 10 функціональних вимог з боку диспетчера, 10 функціональних вимог з точки зору менеджера автопарку, 10 нефункціональних вимог.

Завдання 9. Розробіть прототипи інтерфейсу для системи «*Диспетчер таксі*».

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

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

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

Завдання 13. Розробіть документ *Vision* для підсистеми графічних елементів САПР радіоелектронної техніки.

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. 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.

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

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

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

Додаток А
Уніфікована мова моделювання

UML – це не візуальна мова програмування, але її моделі прямо транслюються в текст на мовах програмування та утаблиці реляційної БД.

Словник *UML* утворюють три різновиди будівельних блоків: предмети, відносини та діаграми.

Предмети – це абстракції, які є основними елементами в моделі, відносини пов'язують ці предмети, діаграми групують колекції предметів.

А.1 Предмети

У *UML* є чотири різновиди предметів: структурні предмети; предмети поведінки; предмети групування; предмети-коментарі.

Ці предмети є базовими об'єктно-орієнтованими будівельними блоками. Вони використовуються для написання моделей.

Структурні предмети є іменниками в UML моделях. Вони являють собою статичні частини моделі, понятійні або фізичні елементи.

1. ***Клас*** – це опис багатьох об'єктів, що поділяють однакові Властивості, операції, відносини та семантику. Клас реалізує один або декілька інтерфейсів (рис. А.1).

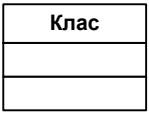

Рисунок А.1 – Клас

Активний клас – це клас чиї об'єкти мають один або декілька процесів (або потоків) і тому можуть ініціювати керуючу діяльність. Активний клас схожий на звичайний клас за винятком того, що його об'єкти діють одночасно з об'єктами інших класів.

2. ***Інтерфейс*** – набір операцій, що визначають послуги класу або компонента (рис. А.2). Інтерфейс описує поведінку елемента, видиму ззовні. Інтерфейс може представляти повні послуги класу або компонента. Інтерфейс визначає набір специфікацій операцій, а не набір реалізацій операцій. Інтерфейс рідко показують самостійно. Зазвичай його приєднують до класу або компоненту, який реалізує інтерфейс.

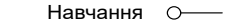

Рисунок А.2 – Інтерфейс

3. ***Кооперація*** визначає взаємодію і є сукупністю ролей та інших елементів, які працюють разом для забезпечення колективної поведінки більш складного, ніж проста сума всіх елементів. (рис. 2.8) Таким чином, кооперації мають як структурне, так і поведінкове вимірювання. Конкретний клас може брати участь в декількох кооперациях. Ці кооперації являють собою реалізацію патернів, які формують систему.

Рисунок А.3 – Кооперація

4. ***Актор*** – набір узгоджених ролей, які можуть грати користувачі при взаємодії з системою. Кожна роль вимагає від системи певної поведінки.

Рисунок А.4 – Актор

5. Елемент ***Use-Case*** (*Варіант використання* або *Прецедент*)– це опис послідовності дій (чи декількох послідовностей), що

виконуються системою в інтересах окремого актора та формують видимий для актора результат (рис. А.5). У моделі прецедент застосовується для структурування предметів поведінки. Елемент Прецедент реалізується кооперацією.

Рисунок А.5 – Прецедент

6. ***Компонент*** – фізична і займана частина системи, що відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів (рис. А.6). У систему включаються як компоненти, які є результатами процесу розробки (файли вихідного коду), так і різні різновиди використовуваних компонентів. Зазвичай компонент – це фізична упаковка різних логічних елементів (класів, інтерфейсів і співробітництв).

Рисунок А.6 – Компонент

7. ***Вузол*** – фізичний елемент, що існує в період роботи системи і являє собою деякий ресурс, який зазвичай має пам’ять і можливості обробки (рис. А.7). У вузлі розміщується набір компонентів, що може переміщатися від одного вузла до іншого.

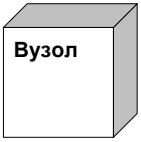

Рисунок А.7 – Вузол

Предмети поведінки – це динамічні частини UML-моделей. Вони є дієсловами моделей, уявою поведінки в часі і просторі. Існують дві основні різновиди предметів поведінки.

1. Взаємодія – поведінка, що містить у собі набір повідомлень, якими обмінюється набір об'єктів в конкретному контексті для досягнення певної мети (рис. А.8).

Взаємодія може визначати динаміку як сукупності об'єктів, так іокремої операції. Елементами взаємодії є повідомлення, які єпослідовністю дій та зв'язку.

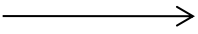

Рисунок А.8 – Взаємодія

2. Кінцевий автомат – поведінка, яка визначає послідовність станів об'єкта або взаємодії, що виконуються в ході його існування увідповідь на події (рис. А.9). З допомогою кінцевого автомата може визначатися поведінка індивідуального класу або кооперації класів. Елементами кінцевого автомата є стани, переходи (від стану до стану), події (предмети, що викликають переходи) та дії (реакції на перехід).

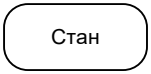

Рисунок А.9 – Стан

Ці два елементи взаємодії і кінцеві автомати – є базисними предметами поведінки, які можуть включатися в *UML* моделі. Семантично ці елементи асоціюються з різними структурними елементами.

Предмети групування – організаційні частини *UML* моделей. Це шухляди, по яких може бути розкладена модель.

Пакет – загальний механізм для розподілу елементів за групами. У пакет можуть поміщатися структурні предмети, предмети поведінки і навіть інші групування предметів. На відміну від компонента, пакет– чисто концептуальне поняття. Це означає, що пакет існує тільки вперіод розроблення (рис. А.10).

Рисунок А.10 – Пакет

Предмети-коментарі – роз'яснюють частини *UML*-моделей. ***Примітка*** – символ для відображення обмежень і зауважень, приєднуваних до елементу або сукупності елементів.

Рисунок А.11 – Примітка

А.1.1 Відношення в *UML*

UML використовує чотири різновиди відношень:

– залежність;
– асоціацію;
– узагальнення;
– реалізацію.

Ці відносини є базовими будівельними блоками відносин. Вони використовуються при написанні моделей.

1 ***Залежність*** – семантичне відношення між двома предметами, в якому зміна в одному предметі може впливати на семантику іншого предмета (рис. А.12).

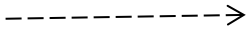

Рисунок А.12 – Відношення залежності

2. ***Асоціація*** – структурне відношення, що описує набір зв'язків, які є з'єднанням між об'єктами (рис. А.13).

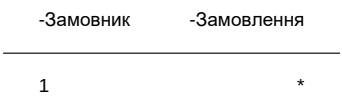

Рисунок А.13 – Асоціація

3. ***Узагальнення*** – відношення спеціалізації/узагальнення, вякому об'єкти спеціалізованого елемента (нащадка) можуть замінювати узагальненений елемент (предка) (рис. А.14).

Рисунок А.14 – Відношення узагальнення

4. ***Реалізація*** – семантичне відношення між класифікаторами, де класифікатор визначає контракт, який другий класифікатор зобов'язується виконувати (рис. А.15). Відносини реалізації застосовують у двох випадках: між інтерфейсами і класами (або компонентами), що реалізовують їх; між прецедентами і кооперації, які реалізують їх.

Рисунок А.15 – Відношення реалізації

А.1.2 Механізм розширення в *UML*

UML – розвинута мова з великими можливостями, але навіть вона не може відбити всі нюанси, що можуть виникнути при створенні різних моделей. Тому *UML* містить засоби розширення:

– обмеження;
– тегові величини;
– стереотипи.

Обмеження (*constraint*) – розширює семантику будівельного *UML*-блоку, дозволяючи додати нові правила або модифіковані існуючі. Обмеження показують як текстовий рядок, укладений уфігурні дужки {} (рис. А.16).

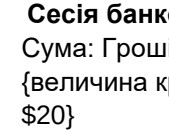

Рисунок А.16 – Використання обмежень на атрибути

Тегова величина (*tagged value*) – розширює характеристики будівельного *UML*-блоку, дозволяючи створювати нову

інформацію вспецифікації конкретного елемента. Тегові величини показують як рядок у фігурних дужках {}. Рядок має вигляд

ім’я тегової величини = значення;

Стереотип дозволяє розширювати словник мови, дозволяє створювати нові види будівельних блоків, похідні від існуючих та враховують специфіку нової проблеми. Елемент зі стереотипом єваріацією існуючого елемента, що має таку ж форму, але відрізняється по суті. Відображають стереотип як ім’я в подвійних кутових дужках.

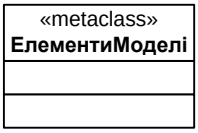

Рисунок А.17 – Приклад використання стереотипів