Лабораторна робота 2

Тема: Архітектура програмного забезпечення

Мета: отримати практичні навички в обґрунтованому виборі компонентів архітектури програмного забезпечення для реалізації завдання та його нефункціональних властивостей.

Розробники програмного забезпечення постійно вишукують прозорі та досконалі архітектури для своїх проектів тому, що це обумовлює отримання реалізації без помилок та дефектів, а також забезпечує повторне використання. Архітектура програмного забезпечення це опис на рівні його компонентів та зв'язків між ними. Відповідно до стандарту IEEE1471 архітектура ПЗ є базовою організацією системи, яку втілено в її компонентах, їх відношеннях між собою та оточенням, а також принципи, що визначають проектування та розвиток системи.

Створення архітектури представляє собою проектування на найвищому рівні. Наступною за нею частиною буде процес детального проектування. Прозорий опис архітектури є важливим для усіх типів розробок і є обов'язковим коли у розробці приймає участь значна кількість людей. Однією з причин цього є необхідність поділу всього додатку на частини (модулі) з наступною їх інтеграцією. Вибір архітектури забезпечує потрібну модульність. Зазвичай у команді розробників є архітектор (технічний архітектор) або його роль періодично виконує найбільш досвідчений у команді фахівець.

Для конкретного проекту розробки програмного забезпечення може підходити декілька архітектур, із яких необхідно обрати найкращу. Зазвичай складно задовольнити усі вимоги однаково ефективно тому, що різні архітектури можуть краще підходити для виконання однієї вимоги і слабкіше або зовсім не виконувати іншу. Саме тому усі вимоги мають певні пріоритети. До властивостей, які слід враховувати при виборі архітектури програмного забезпечення, відносяться:

- розширення — здібність додавання нових властивостей ПЗ;
- здатність до змін — здібність до зміни вимог;
- простота — простота розуміння та простота реалізації;
- ефективність:
  - досягнення високої швидкості виконання та компіляції;
  - досягнення малого розміру коду для моделі даних та реалізації алгоритму.

Розширення визначає ступінь підтримки архітектурою додавання нових можливостей у додаток. Чим краще архітектуру пристосовано до розширення, тим більш складну структуру вона має і тим більше часу буде потрібно на її розробку. Здатність до розширення зазвичай вимагає більш високих абстракцій у процесі.

Безумовно, що універсальність надає значні переваги, але її реалізація потребує і великих затрат часу. Однією з важливих задач при обрані ступені універсальності є визначення класу можливих розширень. Наприклад, можна визначити вимогу, що програмне забезпечення можна було б застосувати для створення будь-якого Інтернет-магазину, але це, швидше за все, можна буде реалізувати не створенням Інтернет-магазину, до якого внесенням змін можна було б досягти бажаних властивостей, а деякої платформи, за допомогою якої можна було б створювати бажаний ресурс. Тому корисними є і необов'язкові, і бажані вимоги, які у сукупності впливають на архітектуру та напрямок розвитку додатку.

Розробка із розрахунком на зміни має на увазі дещо іншу мету, але застосовує ті самі засоби проектування, що і для забезпечення розширення. Наприклад, при розробці деякої комп'ютерної гри є вимога, що гравець постійно керує та має постійний контроль над певним персонажем. Але, наприклад, сценарист такої гри вказує на те, що має і трохи інший, але не затверджений сценарій, у якому гравець може одночасно контролювати декілька персонажів. Саме тому у розробку архітектури слід включити вимогу щодо можливих змін властивостей по відношенню до контролю та керуванню персонажем.

Простота, прозорість достатньо часто є додатковою вимогою у різних проектах та за різних обставин. Прозора та зрозуміла архітектура, що допускає можливість широкого спектру розширень та змін, є великою рідкістю, а її створення потребує великих зусиль. До інших критеріїв, що беруть до уваги при обрані архітектури, відносяться економія машинного часу та пам’яті, насамперед оперативної.

Принциповою проблемою програмного забезпечення є його складність, яка обумовлюється щільністю взаємозв’язків. Методом, який дозволяє це подолати, є декомпозиція або модульність задачі, тобто розбивання задачі на підзадачі, які б можна було б вирішити за допомогою менших частин виконуваного коду. Насамперед виконується відокремлення підзадач із загальної задачі додатку (програмної або інформаційної системи), а потім виконується декомпозиція підзадач, тощо. Таке проектування називається рекурсивним і має за мету визначити атомарні частини коду, зв’язки між ними та передує інтеграції атомарних частин у логічно і технічно зрозумілі компоненти або модулі.

Зв’язність усередині модуля — це сила взаємозв’язку між елементами модуля. Зчеплення характеризує ступінь взаємодії модуля з іншими модулями. Ефективна модульність досягається максимізацією зв’язності та мінімізацією зчеплення. Такий підхід дає можливість поділяти складні задачі на більш прості. Мале зчеплення у сукупності зі значною зв’язністю є достатньо важливими при проектуванні програмного забезпечення, що обумовлюється постійним процесом внесення змін до проектів. Архітектури з малим зчепленням і значною зв’язністю більш пристосовані до модифікації тому, що зміни в таких архітектурах значно локалізуються. Проблеми та складність декомпозиції можна продемонструвати на прикладі персонального фінансового додатку. Достатньо прозора для користувача декомпозиція цього додатку може виглядати наступним чином:

- Рахунки (перевірка, зберігання, тощо);
- Сплата рахунків (карткою, через термінал, готівкою, тощо);
- Глобальні звіти (загальні активи, заборгованість, тощо);
- Кредити (автомобіль, освіта, житло, тощо);
- Інвестиції (акції, позики, тощо).

Але з точки зору архітектурної декомпозиції вона має значні недоліки. Наприклад, модуль “Рахунки” має слабку зв’язність тому, що рахунки слабко взаємозв’язані. При цьому зчеплення модулів достатньо велика. Наприклад, для визначення заборгованості необхідно задіяти рахунки, сплату рахунків, кредити і звіти.

Можливою альтернативою декомпозиції такого додатку може бути:

- Інтерфейс (інтерфейс користувача, комунікаційний інтерфейс, тощо);
- Постачальники (орендодавець, кредити, комунальні послуги, тощо);
- Активи (банківські рахунки, депозити, акції власника, майно, тощо).

Таким чином, архітектура програмного забезпечення (ПЗ) представляється певною множиною основних проектних рішень щодо організації програмного забезпечення. Вона, за суттю, є планом розробки майбутнього програмного рішення, а також основою для подальшого життєвого циклу ПЗ. Проектні рішення охоплюють всі аспекти розробки: набір структурних елементів та їх інтерфейсів, їх поведінку та взаємодію з іншим елементами та ПЗ, компонування елементів у підсистеми та компонування усієї системи, а також стіль архітектури та нефункціональні властивості. «Основний» означає ступінь важливості, який надає певному проектному рішенню статус архітектурного. Тобто не всі проектні рішення є архітектурними і саме тому вони не впливають на архітектуру. Важливість проектних рішень залежить від цілей, які необхідно досягнути завдяки розробці ПЗ. Аспект тимчасовості архітектури означає, що у будь-який момент часу ПЗ має тільки одну архітектуру і вона буде змінюватися з часом.

Розпорядча системна архітектура складається з проектних рішень, що були прийняті перед конструюванням ПЗ. Також її можна назвати продуманою чи призначеною архітектурою. Описова архітектура ПЗ визначає те, як систему було побудовано. Це реалізована чи введена у використання архітектура. Під час змін у ПЗ розпорядча архітектура

змінюється першою. На практиці під час змін ПЗ, змінюють її описову архітектуру.

Архітектурний дрейф – це введення основних проектних рішень до описової архітектури ПЗ, які не включені до розпорядчої архітектури, але які не конфліктують з її рішеннями. Архітектурна ерозія – це введення архітектурних рішень до описової архітектури ПЗ, при цьому рішення не повинні порушувати його розпорядчої архітектури.

Відновлення архітектури – це процес відтворення архітектури на основі компонентів ПЗ, що були отримані на етапі її конструювання. ПЗ не може виконувати свого призначення до того, як воно буде розгорнуте та налаштоване. Модулі ПЗ фізично розміщують на пристроях, на яких вони будуть виконуватися. Архітектурне подання розгортання часто є критичним для оцінювання чи буде система відповідати вимогам. Можливі критерії оцінки: використання пам’яті, споживання енергії, вимоги до пропускної спроможності мережі та інші.

Архітектура ПЗ має бути композицією елементів, котрі репрезентують обробку інформації, інформацію, та комунікацію. Компонент – це архітектурна одиниця, яка відповідає за функціональність ПЗ та/або зберігання даних, надає доступ до своїх сервісів за допомогою зовнішнього інтерфейсу і має чітко визначену залежність від контексту у якому цей компонент застосовують. У складних системах взаємодія між компонентами є більш важливою, ніж функціональність окремих частин ПЗ.

З’єднувач – це архітектурна одиниця, що відповідає за здійснення взаємодії між компонентами. У багатьох системах з’єднувачі репрезентовані звичайним викликом процедур, або спільним доступом до даних, але в цілому з’єднувачі можуть бути більш складними рішеннями. Компоненти та з’єднувачі, скомпоновані у спеціальному порядку, складають готову архітектуру ПЗ. Архітектурна конфігурація – це специфікація з’єднань між з’єднувачами та компонентами.

Нефункціональні властивості системи (НВС) – це обмеження на те, як ПЗ реалізує і доставляє свою функціональність. Наприклад: ефективність, складність, здатність до розширення, надійність. Будь-який високотехнологічний продукт характеризується своїми функціональними можливостями. Забезпечення необхідної функціональності часто є досить складним завданням через потреби ринку, конкуренцію, жорсткі терміни, обмежені бюджети, тощо. Однак успіх системи цілковито залежить від НВС. Роль архітектури – це забезпечення НВС на рівні архітектурних блоків: компонентів, з’єднувачів, конфігурації. Ефективність – це якість, яка відображає здатність ПЗ до задоволення вимог продуктивності при одночасній мінімізації використання його ресурсів. Складність вказує до якої міри ПЗ або один з його компонентів, містить проектні рішення чи реалізацію, які важко зрозуміти і перевірити. Масштабованість ПЗ – це можливість системи бути зміненою з урахуванням нових вимог. Неоднорідність – це якість ПЗ, що передбачає, що ПЗ складається з декількох різнорідних компонентів або функціонує в декількох різнорідних обчислювальних середовищах одночасно. Пристосовуваність – є здатністю ПЗ до задоволення нових вимог і пристосовуватися до нових умов роботи під час його життєвого циклу. Надійність – це набір властивостей ПЗ, що дозволяє розраховувати, що ПЗ буде функціонувати так, як було заплановано.

Функціональна схема будується з метою розуміння всіх функцій, що виконує ПЗ.

Рис1. - Функціональна схема системи

З наведеної функціональної схеми видно, що програмний засіб розбивається на n основних блоків:

1 - авторизація користувача,

2 - робота з довідковими даними,

3 - робота з даними по командах,

4 - ...

Дані передаються або з БД, або вводяться з клавіатури. Після кожного блоку передбачено перегляд результатів роботи даного блоку. Дані, які змінюються в процесі роботи зберігаються в БД.

При виборі архітектури програмного засобу, як правило, ставляться наступні завдання та вимоги:

- створення структури даних, що чітко відображають специфіку предметної області;
- моделювання реально існуючих процесів;
- забезпечення оптимальності структур даних;
- поділ і угруповання функцій програмного засобу на підзадачі;
- забезпечення максимальної надійності програмного засобу;
- забезпечення функціональної повноти відповідно до постановки завдання;
- мінімізація інформаційних потоків усередині системи, що дозволяє скоротити час обробки інформації;
- забезпечення наочності модельованих процесів шляхом візуалізації.

У модульній структурі з функціональною зв'язністю і низьким зчепленням структури даних і функції вносяться до модулів за функціональною ознакою, що забезпечує реалізацію конкретних підзадач у межах окремого модуля. Даний підхід дозволяє спростити контроль над збереженням цілісності логіки, а так само спрощує супровід і модернізацію програмного комплексу. Низьке зчеплення модулів дозволяє проводити модернізацію і налагодження кожного модуля окремо, а так само проводити розширення функціональності програмного комплексу шляхом створення додаткових модулів і приєднання його в загальну структуру шляхом підключення його до головного модулю.

Все вище перераховане забезпечує значну гнучкість у використанні програмного засобу. Ієрархічну (структурну) схему модулів програмної системи представлено на рис.2.

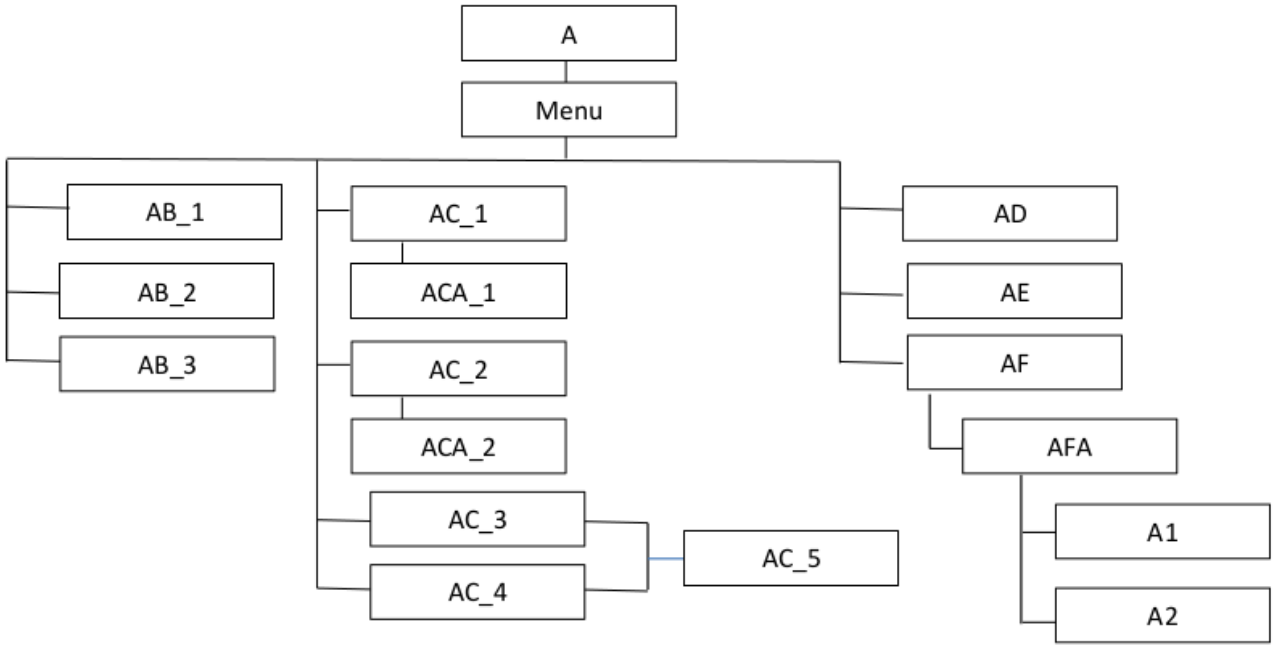

Рис2. – Ієрархічна схема модулів системи

Приклад опису модулів програмної системи:

Модуль A - у цьому модулі відбувається авторизація користувача, якщо авторизація пройшла успішно модуль запускає модуль MENU.

Модуль MENU - основний модуль програмної системи з нього здійснюється зв'язок з іншими модулями програмної системи.

Модуль AB_1 - викликається з модуля MENU для додавання даних. Після своєї роботи модуль передає управління модулю MENU.

Модуль AB_2 - викликається з модуля MENU для зміни даних. Після своєї роботи модуль передає управління модулю MENU.

Модуль AB_3 - викликається з модуля MENU для видалення даних. Після своєї роботи модуль передає управління модулю MENU.

Модуль AC_1 ...

Модуль AC_2 ...

Модуль AC_3 ...

Модуль AC_4 ...

...

В UML для представлення специфікацій загальної структури програмного коду системи використовують діаграму компонентів (рис.3). Діаграма компонентів забезпечує узгоджений перехід від логічного до фізичного представлення системи у вигляді програмних компонентів. Деякі компоненти можуть існувати тільки на етапі компіляції програмного коду,

інші – на етапі його виконання. Основними елементами діаграми є компоненти, інтерфейси та залежності між ними, а також вона може вміщувати ключові класи, що входять до складу компонент.

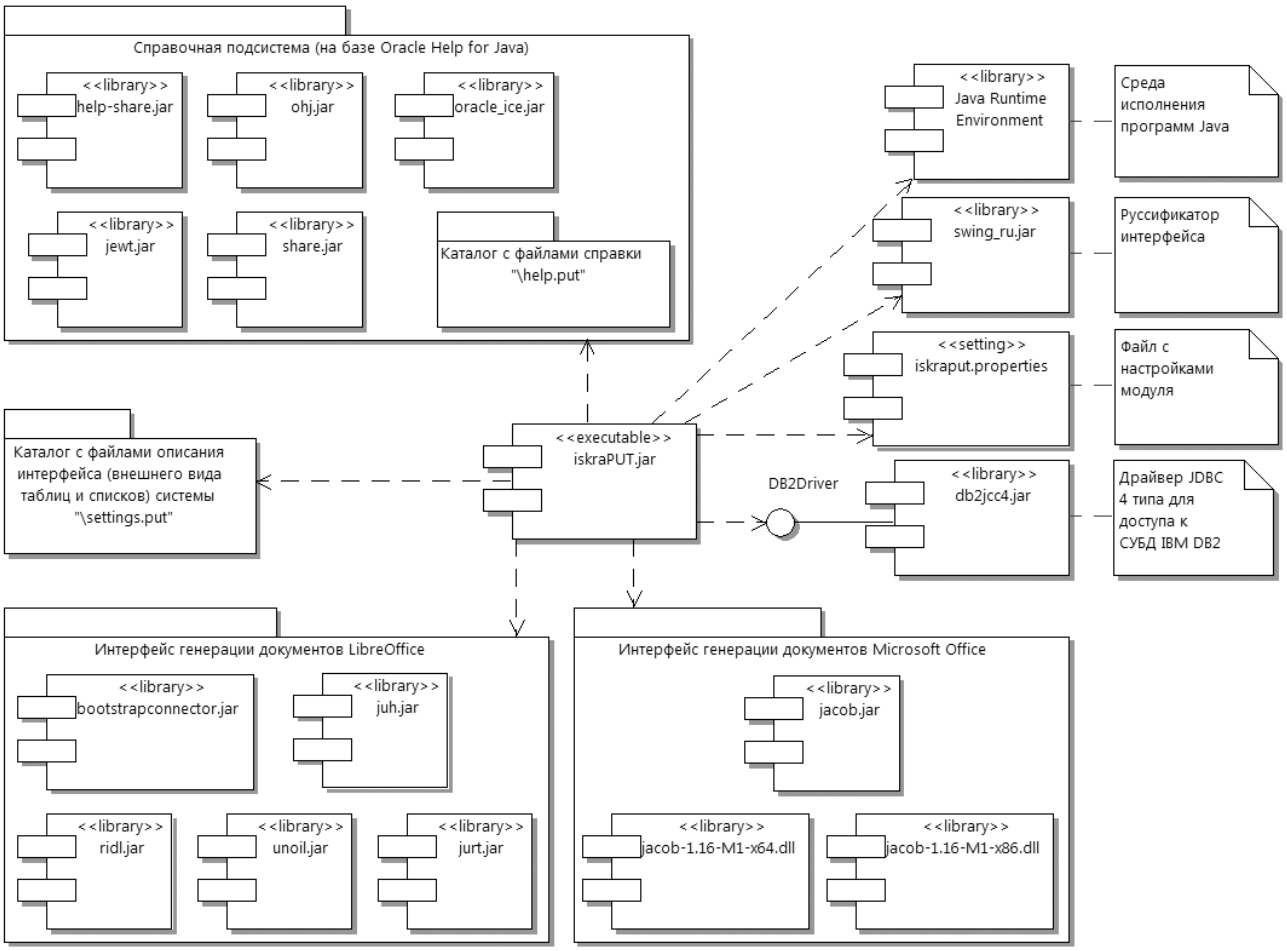

Рис. 3 — Діаграма компонентів відображає структуру програмного коду системи.

Структура є важливою характеристикою архітектури ПЗ, а структурні аспекти архітектури проявляються у різні способи. Структурним елементом може бути підсистема, процес, бібліотека, база даних, тощо. Достатньо часто структурні елементи представляють за допомогою діаграми класів UML (рис.4).

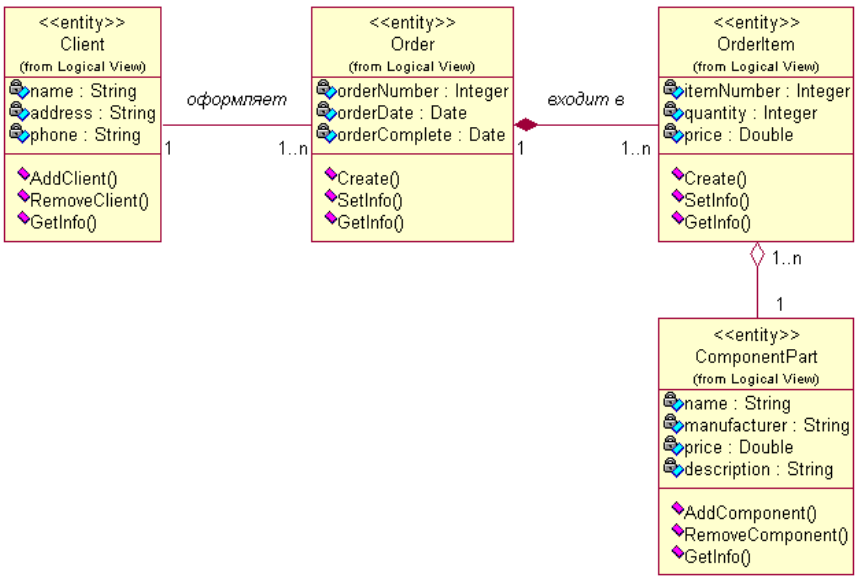

Рис.4 — Діаграма класів часто є концептуальною моделлю структурних елементів.

Крім визначення структурних елементів будь-яка архітектура визначає взаємодію між цими структурними елементами, що забезпечує бажану поведінку системи. На рис.5 представлено діаграму послідовностей UML, яка показує декілька взаємодій, що дозволяють системі підтримувати створення замовлення в модулі обробки замовлень. На діаграмі представлено п’ять взаємодій. Актор Sales Clerk створює замовлення за допомогою екземпляру класу OrderEntry. Екземпляр класу OrderEntry отримує відомості про клієнта за допомогою екземпляру класу CustomerManagement. Екземпляр класу OrderEntry використовує екземпляр класу AccountManagement для створення замовлення, додавання до нього елементів замовлення і розміщення замовлення.

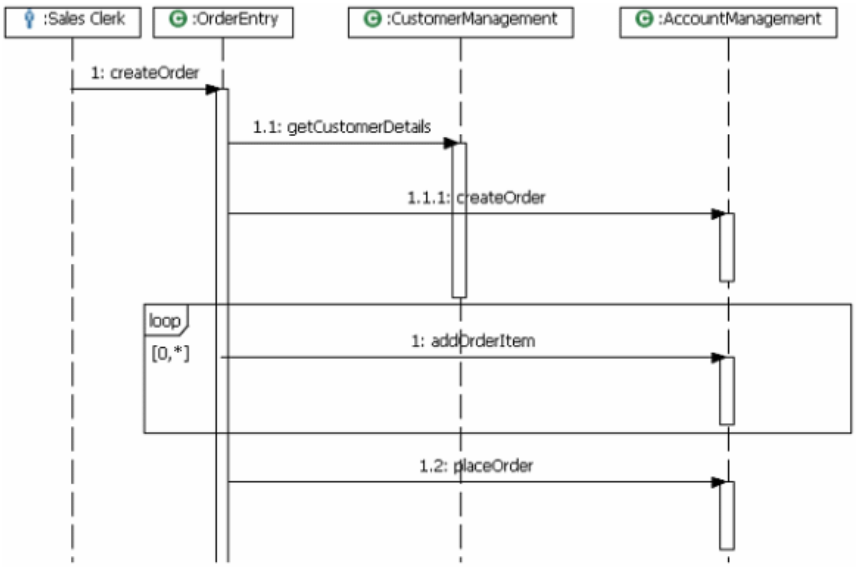

Рис. 5 — Діаграма послідовностей представляє поведінку системи за певною функцією.

Завдання

1. Опрацюйте теоретичний матеріал до лабораторної роботи.
2. Визначте основні функціональні та нефункціональні вимоги, які необхідно реалізовувати у своєму проекті.
3. Побудуйте функціональну та структурну схему архітектури ПЗ.
4. Представте діаграму компонентів для системи.
5. Представте один структурний елемент у вигляді діаграми класів.
6. Представте поведінку системи за однією функцією або однією задачею діаграмою послідовностей.
7. Оберіть шаблон (шаблони) проектування, які задовольняють вимогам ПЗ та обґрунтуйте цей вибір.
8. Покажіть компоненти та з’єднання, які будуть використовуватися та надайте обґрунтування.
9. Покажіть як буде змінюватись архітектура ПЗ та типи шаблонів проектування, що будуть застосовуватись для реалізації додатку при зміні пріоритетів таких необов’язкових вимог, як розширення, здатність до змін, простота та ефективність.
10. Підготуйте звіт.

Рекомендована література

1. Гагарина Л.Г. Технология разработки программного обеспечения. - М.:ИД «Форум» - Инфра-М, 2008.- 408 с.
2. А.Рудаков Технология разработки программных продуктов М.: ИЦ "Академия", 2013р. - 108 с.

Контрольні запитання

1. Що таке архітектура ПЗ?
2. Що таке описова та розпорядча архітектура ПЗ?
3. Що таке архітектурний дрейф та ерозія?
4. Що таке відновлення архітектури?
5. Що таке компонент ПЗ?
6. Для чого використовується діаграма компонентів?
7. Яка роль діаграм класів у проекті?
8. Для чого призначено діаграму послідовностей?
9. Що таке з’єднувач ПЗ?
10. Що таке архітектурна конфігурація або топологія?
11. Що таке нефункціональна властивість ПЗ?
12. Що таке ефективність ПЗ?
13. Що таке складність ПЗ?
14. Що таке надійність ПЗ?
15. Що таке адаптованість ПЗ?