МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ  
«ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ЛУЦЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ»

Випускова циклова комісія Компʼютерних систем та інформаційних технологій

**Звіт**

**з практичної роботи №6**

з дисципліни “Технології розробки програмного забезпечення”

Виконав: студент **2** курсу, групи **22-ІСТ**

**Синиця Назар**

Викладач: Володимир Остапюк

Термін здачі: 20-11-2024

Здано:

Луцьк – 2024

**Завдання**

**Дослідити принципи та підходи DRY, KISS, YAGNI та дати розгорнуті відповіді на запитання:**

1. Що означають принципи DRY, KISS, YAGNI?
2. Які переваги застосування принципів DRY, KISS, YAGNI?
3. Які наслідки можуть виникнути при ігноруванні цих принципів?
4. Як принципи DRY, KISS, YAGNI пов'язані між собою?
5. Як визначити, чи порушено принцип DRY, KISS, YAGNI у коді?
6. Як застосовувати принципи DRY, KISS, YAGNI на практиці?
7. Як ці принципи впливають на процес розробки програмного забезпечення?
8. Як ці принципи впливають на якість коду та його супроводження?
9. \*Які інструменти та методи можуть допомогти у дотриманні цих принципів?

**Висновок**

**Принципи DRY, KISS, YAGNI?**

### DRY — Don't repeat yourself ( Не повторюйтеся )

### Коли ми створюємо певний функціонал, нам доводиться писати схожі або ж однакові методи та класи увесь час. Уявіть, що ви написали метод, який трансформує строку з URL в певний читабельний user-friendly вигляд і ви продублювали її через весь додаток. В якийсь момент хтось каже, що замість підкреслення в посиланні хоче бачити двокрапку. У такому разі вам доведеться пройти через весь додаток і виправити це. Це займе багато часу. Крім того, змінюючи код, можна щось випадково зламати. За принципом DRY ви повинні не просто писати один і той же код один раз, а уникати повторення знання.

### Якщо в якомусь місці ви бачите, що такий код вже був написаний, це момент, коли його треба відрефакторити. Тобто варто подивитись, що тут спільного. Повністю спільне можна винести в reusable клас, компонент або метод.

### Основна задача — зменшити кількість знань, що повторюється. Не повністю прибрати його, а саме зменшити його кількість. Частина коду має бути реалізована лише в одному місці. Також не варто забувати про SRP (Single Responsibility Principle), коли кожний модуль має робити щось одне. Про це поговоримо трохи пізніше.

### Що ж робити, якщо при спробі не повторювати код, ви бачите, що порушуєте принцип DRY? Можливо, краще повторити код у двох місцях, але при цьому Single Responsibility принцип не порушити. Або ж можна розділити код на дві або три частини та використати ці частини окремо. Все залежить від ситуації.

1. **KISS — Keep it simple, stupid ( Будь простіше, дурний )**

Основна ідея принципу KISS — чим простіше, тим краще, щоб кожен розробник зміг зрозуміти, що відбувається у класах та методах. Звідси випливає два тезиси:

* *Keep it short and simple — коротко і просто.*
* *Keep it simple and straightforward — просто і зрозуміло.*

Хороший розробник той, який може написати одну і ту ж логіку так, щоб це було зрозуміло всім. Якщо ж ви створюєте логіку, яка зрозуміла лише вам, то знайте, тут щось не так. Крім того, принцип KISS використовується не лише в програмуванні. Будь-який user experience націлений на простоту — без зайвих дій та перешкод для користувача. Така ж ситуація, наприклад, з автомобілями, телефонами та в цілому в дизайні — простота та зрозумілість.

### YAGNI — You aren't gonna need it ( Вам це не знадобиться )

### Ми багато говорити про те, що варто думати, як ваша програма буде працювати в майбутньому, як вона буде змінюватись, написати код, який легко розширювати і т.д. Але варто не забувати й про принцип YAGNI.

Перед тим, як щось реалізувати, треба подумати, а чи потрібно воно мені зараз. Так, ви можете все продумувати, вирішувати як все буде у майбутньому, але коли ви сідаєте реалізовувати певну задачу, вона має виконувати рівно те, що вам треба в цей момент часу. Не треба створювати методи, які можуть бути потрібними в майбутньому.

90% коду, який пишеться на майбутнє, не працює. І з часом він лише погіршує розуміння вашого коду, бо інші не розуміються ваші методи та для чого вони потрібні. Бізнес-вимоги можуть змінюватись досить швидко, і те, що здається потрібним, завтра вже може бути не актуальним

## Переваги застосування принципів DRY, KISS, YAGNI

Ці принципи є фундаментальними в програмуванні, оскільки вони допомагають створювати код, який є не лише функціональним, але й легким для розуміння, підтримки та розширення. Ось детальніший огляд переваг кожного з них:

### DRY:

* **Зменшення кількості помилок:** Коли логіка міститься в одному місці, ризик ввести помилку при зміні цієї логіки значно зменшується.
* **Покращення читабельності:** Код стає більш структурованим, оскільки повторювані фрагменти винесені в окремі функції або модулі.
* **Спрощення підтримки:** Зміни вносяться швидше, адже потрібно змінити лише одне місце.
* **Підвищення модульності:** Код стає більш незалежним, що полегшує його тестування та повторне використання.

### KISS:

* **Зрозумілість:** Простий код легше зрозуміти для інших розробників, що полегшує співпрацю.
* **Надійність:** Менше складності означає менше потенційних точок збою.
* **Швидкість розробки:** Прості рішення швидше реалізувати.
* **Гнучкість:** Прості системи легше змінювати та адаптувати до нових вимог.

### YAGNI:

* **Економія часу:** Ви не витрачаєте час на розробку функціоналу, який може ніколи не знадобитися.
* **Зменшення технічного боргу:** Менше непотрібного коду, який потрібно підтримувати.
* **Підвищення фокусу:** Ви зосереджуєтесь на вирішенні актуальних задач.
* **Гнучкість:** Система залишається більш гнучкою, оскільки не обтяжена зайвими функціями.

## Наслідки ігнорування принципів DRY, KISS та YAGNI:

Ігнорування принципів DRY, KISS та YAGNI призводить до низки негативних наслідків, які можуть серйозно вплинути на якість програмного продукту, ефективність розробки та загальну продуктивність команди.

### Технічні наслідки:

* **Збільшення технічного боргу:** Дублювання коду, надмірна складність, застарілі рішення створюють технічний борг, який з часом стає все важчим для обслуговування.
* **Зниження якості коду:** Порушення цих принципів веде до зниження читабельності, зростання кількості помилок, меншої модульності та складності в підтримці.
* **Сповільнення розробки:** Більше часу витрачається на дебагінг, рефакторинг та пошук помилок, що призводить до затримок у розробці нових функціональних можливостей.
* **Ускладнення тестування:** Складний код важче тестувати, що збільшує ризик виявлення помилок на пізніх стадіях розробки.

### Організаційні наслідки:

* **Ускладнення співпраці:** Різні стилі кодування, конфлікти при змінах, труднощі з передачею знань ускладнюють роботу в команді.
* **Збільшення витрат:** Виправлення помилок, пов'язаних з порушенням цих принципів, потребує додаткових ресурсів і часу.

**Як принципи DRY, KISS, YAGNI пов'язані між собою.**

DRY і KISS: Дотримання принципу DRY сприяє досягненню KISS, оскільки усунення дублювання коду робить систему простішою.

YAGNI і KISS: YAGNI також сприяє простоті, оскільки ви уникаєте додавання непотрібної складності.

Загалом, ці принципи взаємодоповнюють один одного: DRY забезпечує структуру та повторне використання коду, KISS гарантує простоту та зрозумілість, а YAGNI допомагає зосередитися на найважливіших аспектах.

**Як застосовувати принципи DRY, KISS, YAGNI на практиці?**

Ці принципи, хоч і звучать абстрактно, мають конкретне застосування в повсякденній роботі розробника. Розглянемо, як їх можна застосовувати на практиці:

### DRY (Don't Repeat Yourself) - Не повторюй себе

* Використання функцій та методів: Об'єднуйте повторювані блоки коду в окремі функції або методи. Наприклад, якщо вам потрібно кілька разів перевірити, чи рядок є пустим, створіть для цього окрему функцію.
* Створення констант: Для загальних значень, таких як URL-адреси, кольори, використовуйте константи. Це спростить зміну цих значень в майбутньому.
* Шаблони проектування: Використовуйте готові шаблони проектування, які дозволяють вирішувати типові завдання повторюваним чином.

### KISS (Keep It Simple, Stupid) - Будь простим

* Вибір простих алгоритмів: Якщо є кілька способів вирішити задачу, обирайте найпростіший і найзрозуміліший.
* Уникання надмірної абстракції: Не створюйте зайвих класів та інтерфейсів, якщо вони не необхідні для вирішення задачі.
* Фокусування на основній функціональності: Не намагайтеся реалізувати всі можливі сценарії відразу. Спочатку зосередьтеся на основній функціональності, а потім розширюйте її.

### YAGNI (You Aren't Gonna Need It) - Тобі це не знадобиться

* Відкладання розробки: Не реалізуйте функціонал, який не потрібен зараз. Відкладіть його на потім.
* Мінімалістичний підхід: Створіть мінімально необхідний набір функцій для вирішення поточної задачі.
* Фокусування на користувацьких історіях: Розробляйте функціонал виходячи з конкретних потреб користувачів.

### Приклад: Розробка простого калькулятора

* DRY: Створіть функції для виконання арифметичних операцій (додавання, віднімання тощо), щоб не повторювати код в різних місцях.
* KISS: Використовуйте простий інтерфейс користувача з мінімальним набором кнопок.
* YAGNI: Не реалізуйте складні математичні функції (наприклад, інтеграли), якщо вони не потрібні для основної функціональності калькулятора.

## Як принципи DRY, KISS, YAGNI впливають на процес розробки програмного забезпечення

Принципи DRY, KISS та YAGNI мають фундаментальний вплив на весь процес розробки програмного забезпечення, від початкового етапу планування до підтримки та розвитку продукту. Ось як вони впливають на кожну стадію:

### На етапі планування та проектування:

* Спрощення архітектури: Принципи KISS та YAGNI спонукають розробників створювати простішу та більш гнучку архітектуру, уникаючи надмірної складності.
* Чітке визначення вимог: YAGNI допомагає фокусуватися на найважливіших функціональних можливостях, відкладаючи менш пріоритетні на потім.
* Модульне проектування: DRY сприяє створенню модульних систем, де кожна частина відповідає за свою специфічну функцію.

### На етапі написання коду:

* Повторне використання коду: DRY стимулює розробників створювати функції, класи та модулі, які можуть бути використані в різних частинах програми.
* Проста та зрозуміла логіка: KISS спонукає до написання коду, який легко зрозуміти навіть для інших розробників.
* Мінімалістичний підхід: YAGNI допомагає уникати надмірного коду та спрощує процес написання.

### На етапі тестування та дебагування:

* Швидше виявлення помилок: Простий код легше тестувати, а відсутність дублювання коду зменшує кількість місць, де можуть ховатися помилки.
* Ефективне усунення проблем: Завдяки модульності, знайти та виправити помилку стає простіше.

### На етапі підтримки та розвитку:

* Зменшення технічного боргу: Дотримання цих принципів допомагає зменшити технічний борг, оскільки код стає більш підтримуваним та масштабованим.
* Швидка адаптація до змін: Прості системи легше змінювати та доповнювати новими функціональними можливостями.
* Співпраця в команді: Загальний стиль коду, заснований на цих принципах, полегшує співпрацю в команді.

Загалом, дотримання принципів DRY, KISS та YAGNI призводить до створення більш якісного, надійного та підтримуваного програмного забезпечення. Це, в свою чергу, підвищує продуктивність розробників, знижує витрати на розробку та підтримку, а також покращує задоволеність користувачів.

**Які інструменти та методи можуть допомогти у дотриманні цих принципів?**

Дотримання принципів DRY, KISS та YAGNI є фундаментом для створення чистого, підтримуваного та ефективного коду. Для їх реалізації можна використовувати різноманітні інструменти та методики:

### Інструменти для автоматизації та аналізу

* Лінти та форматери: ESLint для JavaScript, Pylint для Python (Функціонал: Автоматично перевіряють код на відповідність встановленим правилам стилю та виявляють потенційні помилки)
* Статичні аналізатори: SonarQube, Code Climate (Функціонал: Глибоко аналізують код, виявляючи складніші проблеми, такі як потенційні баги, вузькі місця, порушення безпеки)
* Інтегровані середовища розробки (IDE): Visual Studio Code, IntelliJ IDEA (Функціонал: Інтегрують лінти, форматери, дебагери та інші інструменти, надаючи розробникам потужні можливості для написання та аналізу коду)
* Системи контролю версій (VCS): Git (Функціонал: Допомагають відстежувати зміни в коді, співпрацювати в команді та повертатися до попередніх версій)

### Методи для забезпечення якості коду

* Code review: Систематична перевірка коду колегами для виявлення помилок, покращення читабельності та забезпечення дотримання стандартів.
* Модульне тестування: Тестування окремих модулів коду для перевірки їх коректної роботи.
* Рефакторинг: Постійне вдосконалення структури коду без зміни його зовнішньої поведінки.
* Патерни проектування: Використання перевірених рішень для типових задач, що покращує повторюваність та підтримуваність коду.
* Принципи SOLID: Забезпечення гнучкості, розширюваності та тестуємості об'єктно-орієнтованого коду.

## 

## Висновок:

Принципи DRY, KISS та YAGNI є незмінними супутниками будь-якого розробника, який прагне створювати якісний та ефективний код. Дотримання цих принципів дозволяє значно спростити процес розробки, підвищити читабельність та підтримуваність коду, зменшити кількість помилок та прискорити процес розробки.

DRY (Don't Repeat Yourself) підкреслює важливість уникнення дублювання коду. Це означає, що кожна частина інформації повинна мати єдине подання в системі. Такий підхід дозволяє зменшити кількість помилок, спрощує модифікацію коду та підвищує його повторюваність.

KISS (Keep It Simple, Stupid) закликає до простоти у розробці. Складні рішення частіше містять помилки і їх важче підтримувати. Тому варто обирати найпростіші рішення, які вирішують поставлену задачу.

YAGNI (You Aren't Gonna Need It) фокусується на реалізації лише необхідних функцій. Це допомагає уникнути передчасної оптимізації та зменшити складність системи.

Мені як майбутньому програмісту дуже знадобляться знання про дані принципи програмування, і я уже розумію їх важливість тому я прикладу максимум зусиль для їх засвоєння та подальшого використання.