сцени. Завдяки цьому підходу досягається не лише покращення точності класифікації, але й зменшення кількості помилок при складанні опису сцени.

Для прикладу було використано зображення з категорії «Дорожній рух», Google Open Images dataset (рисунок 1). Для складання опису сцени спочатку виконано object detection для даного зображення.

На даному зображенні знайдено такі об’єкти, як: person, top, bicycle, car, shoe, car, tire, bicycle helmet, tire, bicycle wheel, footwear. Через просторове відношення об’єктів, що перетинаються, будуються відношення «person», «bicycle helmet», «bicycle» та «bicycle wheel» як семантично залежні частини візуальної фрази.

Інші об'єкти на зображенні, наприклад автомобілі, також можуть бути використані для визначення змісту сцени. Оскільки об’єкти «велосипед» і «людина» перетинаються, що відповідає статистично стійким семантичним сполученням або прототипам, це визначає контекст та підтверджує, що сцена є транспортною, де людина використовує велосипед для пересування.

Таким чином, в результаті визначення візуальних відношень об’єктів, знайдених на зображенні сцени, може бути виділена структура S_1 – «людина на велосипеді» (рисунок 2).

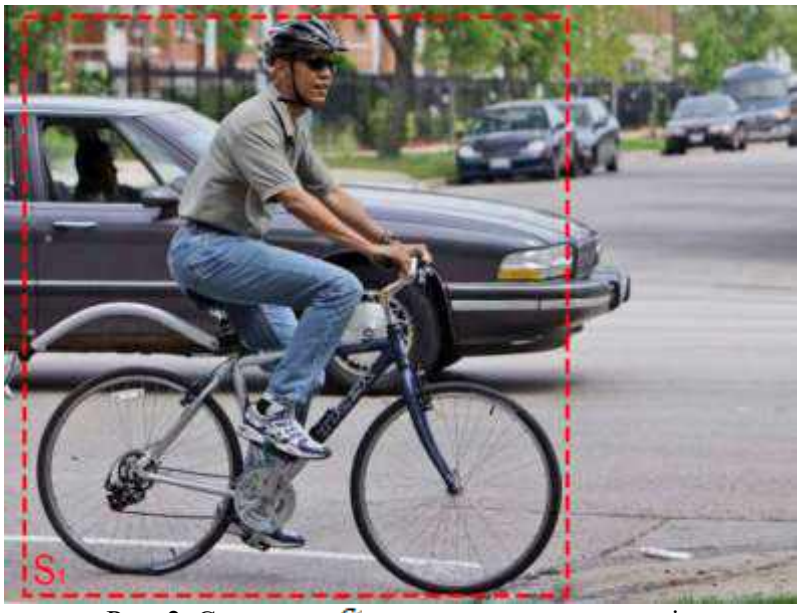

Рис. 2. Структура S_1 – «людина на велосипеді»

Адаптація методу для мобільних пристроїв включає використання трансферних моделей та технологій оптимізації обчислень, таких як грануляція та прототипування візуальних відношень. Це дозволяє досягти швидкодії та забезпечити точність класифікації сцен у реальному часі.

Запропонований метод демонструє високий потенціал для використання у мобільних застосунках, що працюють з медіа-контентом, надаючи користувачам інструмент для інтелектуального аналізу та інтерпретації зображень на основі виявлених відношень між об'єктами.

Список використаних джерел

1. SSD Mobile net [Електронний ресурс] – режим доступу: https://github.com/chuanqi305/MobileNet-SSD. Дата звернення: 04.11.2024р.
2. TensorFlow [Електронний ресурс] – режим доступу: https://www.tensorflow.org. Дата звернення: 04.11.2024р.

ПРУС О.В.,
Вінницький національний технічний університет
МАЙДАНЮК В.П.,
Вінницький національний технічний університет

ЕФЕКТИВНА ВІЗУАЛІЗАЦІЯ ЗАЛЕЖНОСТЕЙ ЯК ЗАСІБ ОПТИМІЗАЦІЇ РОЗРОБКИ У БАГАТОПРОЕКТНОМУ СЕРЕДОВИЩІ

Анотація: Стаття розглядає роль та важливість ефективної візуалізації залежностей як засобу оптимізації розробки у багатопроектному середовищі. Ефективна візуалізація залежностей між

проектами та бібліотеками дозволяє розробникам краще розуміти структуру монорепозиторію, виявляти потенційні проблеми та приймати обґрунтовані рішення щодо архітектури. У статті аналізуються основні аспекти використання інструментів візуалізації залежностей, зокрема NX Dependency Graph, а також надається огляд методів та практик для покращення візуалізації та управління залежностями. Висвітлюються найкращі практики та інноваційні рішення для підвищення ефективності розробки у багатопроектному середовищі.

Ключові слова: візуалізація залежностей, монорепозиторій, багатопроектне середовище, NX Dependency Graph, оптимізація розробки, управління залежностями, інструменти візуалізації, Angular, Nx Monorepo, продуктивність, масштабованість, інтерактивні веб-інтерфейси.

Abstract: The article examines the role and importance of effective dependency visualization as a means of optimizing development in a multi-project environment. Effective visualization of dependencies between projects and libraries allows developers to better understand the structure of a monorepository, identify potential issues, and make informed architectural decisions. The article analyzes the main aspects of using dependency visualization tools, particularly NX Dependency Graph, and provides an overview of methods and practices for improving visualization and dependency management. Best practices and innovative solutions for enhancing development efficiency in a multi-project environment are highlighted.

Keywords: dependency visualization, monorepository, multi-project environment, NX Dependency Graph, development optimization, dependency management, visualization tools, Angular, Nx Monorepo, productivity, scalability, interactive web interfaces.

Вступ. З розвитком сучасних веб-додатків вимоги до масштабованості, підтримки та якості програмного забезпечення зростають з кожним днем. Багатопроектні середовища та монорепозиторії стають все більш популярними серед команд розробників, оскільки вони дозволяють об'єднувати декілька проектів та спільних бібліотек в єдину структуру. Це сприяє повторному використанню коду, уніфікації підходів та спрощує управління залежностями [1, 2].

Проте зі збільшенням кількості проектів та взаємозв'язків між ними зростає складність управління залежностями. Ефективна візуалізація залежностей між проектами та бібліотеками є ключовим фактором для оптимізації процесу розробки у такому середовищі. Вона допомагає розробникам краще розуміти структуру коду, виявляти потенційні проблеми, такі як циклічні залежності або надмірна зв'язаність, та приймати обґрунтовані рішення щодо архітектури [3].

NX Dependency Graph є одним із інструментів, що надають можливості для візуалізації залежностей у Nx Monorepo. Він дозволяє відображати взаємозв'язки між додатками та бібліотеками, що особливо корисно в масштабних проектах на базі Angular та інших фреймворків. Проте, незважаючи на його потужність, існують виклики, пов'язані з масштабованістю, інформативністю та зручністю використання цього інструменту [4].

Актуальність дослідження ефективних методів візуалізації залежностей зумовлена необхідністю підвищення продуктивності та якості розробки у складних проектах. Розробники потребують інструментів, які допомагають їм швидко орієнтуватися у структурі коду, виявляти проблеми та оптимізувати робочі процеси.

Метою цієї статті є дослідження та вдосконалення методів та інструментів для ефективної візуалізації залежностей у багатопроектному середовищі. Ми прагнемо аналізувати сучасні підходи, виявити їхні обмеження та запропонувати інноваційні рішення, які сприятимуть оптимізації процесу розробки та підвищенню ефективності командної роботи.

Аналіз досліджень та постановка завдання

Зі збільшенням масштабів проєктів виникають виклики, пов'язані з ефективним управлінням та візуалізацією залежностей між компонентами. NX Dependency Graph є інструментом, що дозволяє візуалізувати залежності у монорепозиторіях, побудованих з використанням Nx та Angular [4]. Він надає розробникам можливість отримати візуальне представлення структури проєкту, що сприяє кращому розумінню взаємозв'язків між модулями та бібліотеками. Однак, попри наявність офіційної документації та деяких статей від спільноти розробників, детальний аналіз практичного застосування цього інструменту в реальних проєктах все ще є обмеженим.

Основні роботи та публікації з даної тематики включають офіційну документацію Nx, яка надає інформацію про базове використання NX Dependency Graph, його функції та налаштування [5], проте вона не завжди висвітлює глибокі аспекти практичного застосування в масштабних проєктах. Також існують статті від спільноти розробників, де деякі спеціалісти діляться досвідом використання NX Dependency Graph у блогах та на форумах [6][7]. Ці матеріали часто зосереджені на технічних

деталях або окремих випадках використання. Наукові роботи, які аналізують методи управління залежностями у великих проєктах [4], рідко фокусуються на практичному використанні інструментів візуалізації залежностей у контексті монорепозиторіїв.

Невирішеними аспектами, що потребують подальшого дослідження, є практичні кейси застосування, оскільки недостатньо детальних описів реальних проєктів, де використовується NX Dependency Graph з акцентом на практичні результати та вплив на процес розробки. Також потребує дослідження вплив візуалізації залежностей на взаємодію в команді, процес навчання нових членів та ефективність комунікації. Недостатньо інформації про те, як використання NX Dependency Graph допомагає в оптимізації побудови, тестування та впровадження більш ефективних практик CI/CD. Потрібні додаткові практичні приклади, які демонструють, як візуалізація залежностей сприяє виявленню циклічних залежностей, надмірної зв'язаності та інших архітектурних проблем.

Враховуючи наведений аналіз, основними **завданнями** цієї роботи є ознайомлення з практичним застосуванням NX Dependency Graph у реальному масштабному проєкті, побудованому на основі Angular та Nx Monorepo; аналіз впливу використання візуалізації залежностей на процеси розробки, включаючи розуміння структури проєкту, виявлення потенційних проблем, оптимізацію побудови та тестування, а також покращення командної співпраці; визначення переваг та обмежень використання NX Dependency Graph у контексті великого проєкту з метою надання рекомендацій щодо його ефективного впровадження; підготовка практичних рекомендацій для команд розробників, які планують використовувати NX Dependency Graph для покращення управління залежностями у своїх проєктах.

Виклад основного матеріалу

У сучасному світі розробки програмного забезпечення, де складність систем постійно зростає, ефективне управління залежностями стає критичним фактором успіху. Це особливо актуально для монорепозиторіїв, де десятки або навіть сотні проєктів та бібліотек взаємодіють між собою. Розуміння структури цих взаємодій є надзвичайно важливим для забезпечення якості, масштабованості та швидкості розробки [8].

Ефективна візуалізація залежностей покращує розуміння структури проєкту. Вона дозволяє розробникам бачити загальну картину взаємозв'язків між компонентами, що спрощує навігацію по коду та прискорює процес навчання нових членів команди [9]. Нові учасники можуть швидше розуміти архітектуру системи, спостерігаючи за взаємодією різних компонентів. Візуальні діаграми сприяють кращому обговоренню та розумінню складних технічних концепцій у команді.

Візуальні інструменти допомагають ідентифікувати потенційні проблеми, такі як циклічні залежності та надмірна зв'язаність, які можуть негативно впливати на масштабованість та підтримку проєкту [2]. Циклічні залежності можуть призводити до труднощів з побудовою та розгортанням; їх виявлення на ранніх етапах допомагає уникнути складних проблем у майбутньому. Виявлення надмірної зв'язаності дозволяє визначити компоненти з надмірною кількістю залежностей, які важко тестувати та підтримувати.

Знання точних залежностей між проєктами дозволяє оптимізувати процеси розробки, ефективніше планувати роботу, зменшувати час на побудову та тестування, а також впроваджувати більш ефективні практики CI/CD [4]. Розуміння того, які частини системи залежать від змін, дозволяє запускати інкрементальні збірки, економлячи час та ресурси. Це також допомагає фокусуватися на тестуванні лише тих компонентів, які були змінені або можуть бути впливовими.

NX Dependency Graph, інструмент, наданий фреймворком Nx, автоматично генерує граф залежностей між додатками та бібліотеками у репозиторії, надаючи інтерактивний веб-інтерфейс для їх візуалізації [9]. Він дозволяє розробникам взаємодіяти з графом, збільшуючи або зменшуючи масштаб, фільтруючи проєкти та переглядаючи деталі конкретних вузлів. Інтерактивність сприяє швидкому знаходженню необхідної інформації без зайвих зусиль, а інтуїтивний інтерфейс полегшує освоєння інструменту.

Можливість фільтрації та пошуку дозволяє відображати лише певні проєкти або бібліотеки, що спрощує аналіз великих монорепозиторіїв. Це забезпечує таргетований аналіз, фокусуючись на конкретних модулях або командах, і економить час розробників завдяки миттєвому доступу до необхідних компонентів. Відображення залежностей на рівні файлів дає змогу детально дослідити, які саме файли або модулі спричиняють залежність між проєктами. Такий глибокий аналіз допомагає виявити потенційні проблеми, які можуть бути непомітними на вищому рівні, і дозволяє рефакторити код для зменшення непотрібних залежностей [4].

Практичне використання NX Dependency Graph покращує командну співпрацю, оскільки забезпечує всім членам команди доступ до актуальної інформації про структуру проєкту, що сприяє ефективнішій взаємодії. Нові розробники можуть швидше адаптуватися, використовуючи граф залежностей як довідковий матеріал. У плануванні та управлінні проєктами розуміння залежностей допомагає ідентифікувати критичні точки, планувати роботу більш ефективно та проактивно вирішувати потенційні проблеми, що знижує ризик затримок в розробці. Покращення якості коду досягається через рефакторинг, який ідентифікує надмірну зв'язаність та дозволяє оптимізувати кодову базу, а також через дотримання стандартів, де візуалізація допомагає контролювати архітектурні стандарти та патерни проектування [10].

Для успішного впровадження NX Dependency Graph рекомендується інтегрувати його з процесами розробки, включаючи в CI/CD пайплайни для автоматичного оновлення графу залежностей, та використовувати під час код-рев'ю для кращого розуміння впливу змін. Навчання команди через тренінги щодо використання інструменту та інтерпретації графів залежностей, а також створення документації та довідкових матеріалів, забезпечує швидкий доступ до необхідної інформації. Постійний моніторинг та вдосконалення, регулярний аналіз графу залежностей для виявлення та усунення потенційних проблем, і залучення команди до обговорення архітектурних рішень на основі даних з візуалізації сприяють покращенню процесів розробки [8].

Ефективна візуалізація залежностей у монорепозиторіях є потужним засобом для покращення розуміння структури проєкту, виявлення потенційних проблем та оптимізації процесів розробки. Використання таких інструментів, як NX Dependency Graph, надає розробникам інтуїтивний та функціональний спосіб взаємодії зі складними системами, що сприяє підвищенню якості та швидкості розробки. У сучасному багатопроєктному середовищі, де взаємодія між компонентами стає все більш складною, такі інструменти є невід'ємною частиною успішної розробки програмного забезпечення. Впроваджуючи їх у свої процеси, компанії можуть досягти більшої ефективності, покращити співпрацю в командах та забезпечити високий рівень якості своїх продуктів.

Практичне застосування NX Dependency Graph у контексті реального проекту дозволяє виявити переваги та можливості інструменту на практиці та підкреслити його вплив на оптимізацію процесів розробки. У рамках комплексного веб-застосунку, побудованого на основі Angular та Nx Monorepo, виникла потреба в ефективному управлінні залежностями між численними модулями та бібліотеками. Проєкт містить понад 250 модулів, які взаємодіють між собою, утворюючи складну мережу залежностей. На рисунку 1 представлено повний граф залежностей між модулями проєкту (для наочності основну частину модулів було усунуто, відображено 10–12 компонентів).

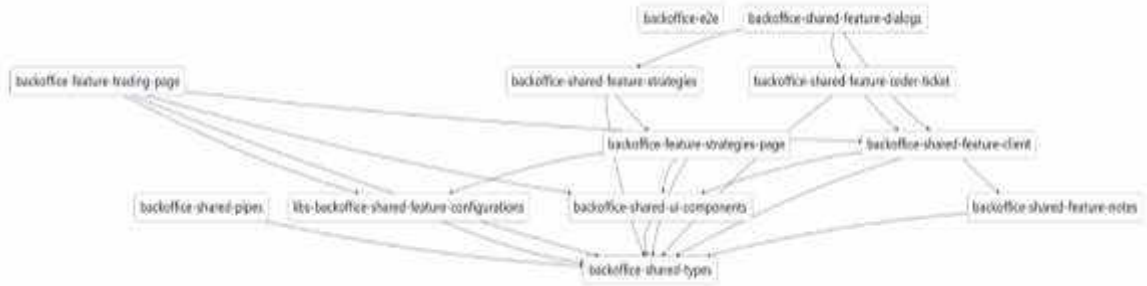

Рис. 1. Граф залежностей між модулями проекту, побудований з допомогою NX Dependency Graph

Кожен вузол відповідає окремому модулю або бібліотеці, а ребра відображають взаємозв'язки між ними. Ця візуалізація дозволила оцінити загальну складність проєкту, зрозуміти, які модулі є центральними, а які менш взаємозалежними; виявити потенційні проблеми, ідентифікувавши кілька модулів з надмірною кількістю вхідних або вихідних залежностей, що могло негативно впливати на масштабованість та тестування; планувати рефакторинг, прийнявши рішення про розділення деяких модулів та оптимізацію їх взаємодії.

Для глибшого аналізу проведено моделювання залежностей у вибраному модулі від інших компонентів системи. На рисунку 2 показано вплив на модуль "backoffice-feature-trading-page" інших частин проекту.

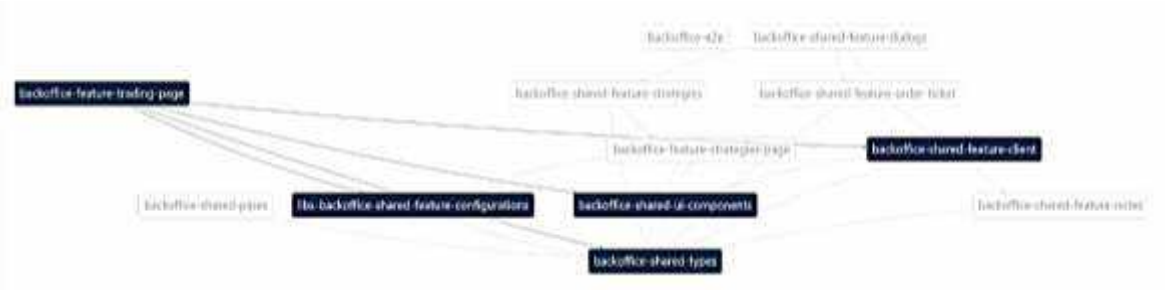

Рис. 2. Моделювання залежностей у вибраному модулі від інших компонентів системи

Аналіз впливу показав, що зміни в модулях "libs-backoffice-shared-feature-configurations", "backoffice-shared-ui-components", "backoffice-shared-types" та "backoffice-shared-feature-client" можуть безпосередньо вплинути на "backoffice-feature-trading-page", оскільки він покладається на їхню функціональність. Опосередкований вплив відзначено на інших модулях, які залежать від "backoffice-feature-trading-page" і можуть зазнати змін через оновлення цих спільних модулів. Велика кількість залежностей вказує на те, що будь-які зміни в зазначених спільних модулях можуть спричинити необхідність перегляду та тестування значної частини системи, особливо "backoffice-feature-trading-page".

На основі аналізу зроблено висновки щодо необхідності модульного тестування для забезпечення стабільності системи, зокрема розробки додаткових тестів для "backoffice-feature-trading-page" та його залежностей. Розглянуто можливість впровадження мікросервісної архітектури з метою зменшення зв'язаності шляхом виділення "backoffice-feature-trading-page" в окремий сервіс або модуль з чітко визначеним інтерфейсом. Відзначено важливість покращення документації, оскільки документування взаємозв'язків допоможе новим членам команди швидше адаптуватися та розуміти вплив своїх змін.

Після проведеного аналізу реалізовано наступні заходи: рефакторинг коду, розділення великих модулів на менші, більш спеціалізовані компоненти, що зменшило кількість прямих залежностей; впровадження абстракцій через використання інтерфейсів та абстрактних класів для зменшення зв'язаності між модулями; автоматизація тестування, налаштування автоматичного запуску тестів для всіх модулів, які впливають на конкретний компонент, за допомогою команд Nx Affected Commands; регулярне оновлення та аналіз графу залежностей як частини процесу розробки [10].

Результати впровадження показали зменшення часу на побудову та тестування на 30% завдяки інкрементальному підходу та оптимізації залежностей. Підвищено стабільність системи, оскільки зміни в одному модулі менше впливають на інші частини проєкту. Покращено співпрацю в команді завдяки кращому розумінню структури проєкту, що зробило комунікацію між розробниками більш ефективною.

Висновки

Практичне використання NX Dependency Graph у проєкті дозволило не лише виявити та усунути поточні проблеми, але й встановити основу для подальшого розвитку системи з урахуванням кращих практик архітектури та управління залежностями. Досягнуто прозорості структури проєкту, покращено якість коду через рефакторинг та впровадження абстракцій, що сприяло зменшенню зв'язаності та підвищенню гнучкості коду. Ефективне управління ризиками стало можливим завдяки аналізу впливу змін, що допомогло мінімізувати ризики при впровадженні нових функцій або змін.

Враховуючи позитивний досвід, планується розробка власних плагінів для Nx з метою автоматизації специфічних для проєкту завдань та покращення інтеграції з іншими інструментами. Передбачається впровадження аналізу історії змін, використання історичних даних для прогнозування потенційних проблем та планування рефакторингу. Заплановано проведення регулярних тренінгів для підвищення компетенції розробників у використанні інструментів візуалізації та управління залежностями.

Таким чином, практичне застосування NX Dependency Graph у реальному проєкті підтвердило його ефективність як засобу оптимізації розробки в багатопроєктному середовищі. Візуалізація залежностей стала невід'ємною частиною процесу розробки, сприяючи підвищенню продуктивності команди та якості кінцевого продукту.

Список використаних джерел

1. Прус О.В., Майданюк В.П., Арсенюк І.Р. Аналіз інструментів управління багатопроєктними середовищами: оптимізація розробки програмного забезпечення. Наукові праці Вінницького національного технічного університету, 2024, 1
2. Прус, О. В., В. П. Майданюк. Багатопроектні середовища та спільна розробка інтерактивних веб-інтерфейсів. Херсонський національний технічний університет, 2023.
3. Juri Strumpflohner. Use the Nx Dependency Graph to Visualize your Monorepo Structure. [Електронний ресурс] – режим доступу: https://egghead.io/lessons/javascript-use-the-nx-dependency-graph-to-visualize-your-monorepo-structure/. Дата звернення: 5 лис. 2024.
4. Enterprise Angular Monorepo Patterns. Nrwl Technologies. Version v0.1, November 26, 2018 [Електронний ресурс] – режим доступу: https://cdn2.hubspot.net/hubfs/2757427/enterprise-angular-mono-repo-patterns.pdf. Дата звернення: 5 лис. 2024.
5. Dheeraj Kumar Rao. An Introduction to Nx: The Ultimate Tool for Monorepos (One tool for almost everything) [Електронний ресурс] – режим доступу: https://medium.com/javascript-kingdom/an-introduction-to-nx-the-ultimate-tool-for-monorepos-one-tool-for-almost-everything-44bd23b203f5/. Дата звернення: 5 лис. 2024.
6. GitHub Discussions forum for nrwl nx [Електронний ресурс] – режим доступу: https://github.com/nrwl/nx/discussions/. Дата звернення: 5 лис. 2024.
7. Juri Strumpflohner Using the Nx Dependency Graph in Custom Scripts [Електронний ресурс] – режим доступу: https://juri.dev/blog/2020/09/use-nx-dep-graph-in-custom-scripts/. Дата звернення: 5 лис. 2024.
8. Explore your Workspace [Електронний ресурс] – режим доступу: https://nx.dev/features/explore-graph /. Дата звернення: 5 лис. 2024.
9. The Nx Dev Tool for Monorepos [Електронний ресурс] – режим доступу: https://dev.to/yggdrasilts/the-nx-dev-tool-for-monorepos-l84/. Дата звернення: 5 лис. 2024.
10. Juri Strumpflohner. Only Build What Changed with Nx Affected Commands [Електронний ресурс] – режим доступу: https://egghead.io/lessons/nx-only-build-what-changed-with-nx-affected-commands/. Дата звернення: 5 лис. 2024.

УДК **004.42**

РЕЙДА М. О.,
СЕРГІЄНКО О. О.,
РЕЙДА О. М.,
Вінницький національний технічний університет

ШАБЛОНИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. Проведено аналіз шаблонів розробки програмного забезпечення.
Ключові слова: Singleton, Factory Method, Facade, Strategy.
Abstract. Analysis of software design patterns has been conducted.
Keywords: Singleton, Factory Method, Facade, Strategy.

Вступ. Шаблони проектування є інструментами для створення структурованого і високопродуктивного програмного коду. В процесі розробки великих проектів команди розробників використовують готові програмні рішення на початкових етапах розробки для зменшення часу розробки і підвищення ефективності. Саме тому дослідження шаблонів є актуальною темою для аналізу задля розробки структурованого і високопродуктивного програмного коду.

Результати дослідження

Шаблони розробки програмного коду відрізняються за складністю, рівнем деталізації та сферою застосування.

Архітектурні шаблони є найбільш універсальними та високорівневими. Розробники можуть використовувати такі шаблони майже будь-якою мовою. На відміну від інших шаблонів, їх можна використовувати для створення архітектури програми.

Шаблони класифікуються за призначенням на такі групи: