УДК 004.415.2 **doi: 10.32620/reks.2021.1.09**

**М. О. БИЧОК, О. К. ПОГУДІНА**

***Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Україна***

# ОЦІНКА ВИКОРИСТАННЯ ШАБЛОНІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

***Предметом*** *вивчення в статті є процеси розробки програмного забезпечення (ПЗ) з використанням патернів проектування.* ***Метою*** *є підвищення якості проектів розробки сучасного ПЗ за рахунок використання досвіду та знань, щодо побудови підсистем ПЗ, які орієнтовані на інфраструктуру та роботу із зовнішнім клієнтом* ***Завдання:*** *огляд методології, парадигм програмування та можливості їх застосування на етапах проектування та кодування життєвого циклу розробки ПЗ; розробка концепції застосування патернів проектування при проектуванні ПЗ як знань, що доступні для повторного використання, запропоновано підхід до практичної реалізації патернів проектування до проєктів node.js. Використовуваними* ***моделями*** *є модель шаблону проектування Composite, модель шаблону проектування Chain of responsibility. Використовуваними методологіями є об'єктно-орієнтоване програмування, як найпоширеніша парадигма програмування сьогодення; уніфікована мова моделювання UML для відображення структури шаблонів проектування Отримані такі* ***результати****. Розглянуто сучасні методології та парадигми проектування, сформовано класифікацію у вигляді деревоподібної структури за поділом на декларативні та імперативні підвиди, зроблено висновок, що у рамках дослідження будемо використовувати об'єктно-орієнтовану методологію, як найпоширенішу парадигму проектування. Розглянуто приклад побудови інформаційної системи проєкту node.js. Проаналізовано основні помилки, що виникають при розробці та написанні коду роботи із зовнішнім клієнтом. Розглянуто елементи проєкту node.js та концепцію структуризації їх взаємозв'язку з існуючими шаблонами проектування. Розглянуто приклад практичної реалізації проєкту node.js та його зв'язок з шаблонами проектування Composite, та Chain of responsibility. У зв'язку з чим у роботі надано структури цих шаблонів.* ***Висновки.*** *Наукова новизна отриманих результатів полягає в наступному: набули подальшого розвитку моделі шаблонів проектування за рахунок їх використання у концепції побудови додатку node.js , що дає можливість підвищити якість взаємодії команди проєкту, скоротити його терміни виконання.*

***Ключові слова:*** *програмне забезпечення; парадигма програмування; шаблони проектування; проєкт node.js.*

## Вступ

На сьогодні розробка програмного забезпечення (ПЗ) важливий процес життєдіяльності людини. Зараз складно уявити життя: без ПЗ, без платіжних систем за допомогою яких купують товари не тільки в крамниці поряд з будинком, а і в іншій країні, без спілкування за допомогою систем відео зв'язку, тощо. Розробка ПЗ давно вийшла за межі наукової та воєнної сфери [1]. Комерційна розробка ПЗ не просто розвивається, а вже займає вагоме місце. Для розробки ПЗ існують певні правила, рекомендації, тобто методологія розробки ПЗ [2]. Кожна методологія характеризується: підходом або основними принципами. Ці принципи, від яких залежить ефективність всієї методології, зазвичай можна коротко сформулювати і легко пояснити; узгодженням множини моделей та методів, які реалізують дану методологію; поняттями, що дозволяють більш точно визначити методи.

В більш вузькому значенні, коли методологія застосовується на стадії програмування (конструювання), її зазвичай називають парадигмою програмування (ППр) [3].

Поширені методології програмування: водоспадна модель (waterfall), макетування (prototyping), ітеративна та інкрементна розробка (iterative and incremental development), спіральна модель (spiral model), швидка розробка ПЗ (rapid application development), екстремальне програмування (extreme programming), різні види методології гнучкої розробки (agile methodology) [2].

Отже існують певні методології які керують або намагаються покращити процес розробки ПЗ. Тобто певні методи, що окутують процеси життєвого циклу ПЗ. Життєвий цикл ПЗ – сукупність окремих етапів

© М. О. Бичок, О. К. Погудіна, 2021

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

робіт, що проводяться у заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення ПЗ і закінчується припиненням використання ПЗ [4]. Можна виділити такі основні етапи розробки ПЗ: аналіз вимог; специфікація ПЗ; проектування ПЗ; програмування; тестування ПЗ; системна інтеграція; впровадження ПЗ (або установка ПЗ); супровід ПЗ. Отже далі буде йти мова про процес проектування та процес програмування.

Вище описані методології мають іноді занадто високий рівень абстракції. Наприклад методологія гнучкої розробки може освітити наступні питання: часті релізи важливі; розмова віч-на-віч – найкращий метод передачі. Проте напряму не вирішує проблеми написання якісного ПЗ. Так відбувається чітка взаємодія між членами команди, менеджментом та замовником, отже програмний продукт (ПП) покращується, але це все ще досить абстрактний вплив.

Тобто, застосування певної методології вимагає формалізації взаємозв’язку з вимогами до ПЗ [5] та практичні реалізації у рамках певної мови та інструментів розробки. Цією ланкою може бути ППр. Таким чином науково-практичною задачею є розробка та використання моделі вибору шаблону проектування при заданих вимогах до ПЗ.

## 1 Огляд парадигм програмування

Метою роботи є підвищення якості проектів розробки ПЗ. Для досягнення мети необхідно провести огляд існуючих методів проектування ПЗ, їх ППр. ППр – це система ідей і понять, які визначають стиль написання комп'ютерних програм, а також спосіб мислення програміста. Це спосіб концептуалізації, що визначає організацію обчислень і структурування роботи, яку виконує комп'ютер. ППр відображає те, як програміст розглядає роботу програми, наприклад: за об'єктно-орієнтованим програмуванням (ООП) – як множини об'єктів, за функціональним програмуванням – як послідовності обчислень функцій без станів. Кожну окрему ППр характеризує наявність у ній методу та зв'язку із моделлю життєвого циклу. Спільним для різних ППр є загальні принципи проектування ПП. Користувач може вибирати ту або іншу ППр з позицій зручності застосування для задач та виготовлення конкретного ПП. Загалом можна виділить два основні підкласи: імперативний та декларативний. Декларативне програмування – ППр, відповідно до якої, програма описує, який результат необхідно отримати, замість описання послідовності отримання цього результату. Наприклад, веб-сторінки HTML – декларативні, оскільки вони описують, що містить сторінка та що має відображатись – заголовок, шрифт, текст, зображення – але не містить інструкцій як її слід відображати. Ця ППр відмінна від імперативних мов, таких як, наприклад, Фортран, C, Java, які вимагають від розробника детального описання алгоритму отримання результатів. Імперативне програмування – ППр, згідно з якою описується процес отримання результатів як послідовність інструкцій зміни стану програми. Можна сказати, що це більш обґрунтовані вимоги до розробки ПП. Тобто використання однієї із ППр дозволяють розробникам більш конструктивно взаємодіяти. Кожна з вище описаних ППр має підвиди (рис. 1). Поточна робота зосереджена на одній із найпопулярніших ППр: ООП – одна з ППр, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність ПЗ (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця ППр з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне ПЗ, що змусило переглянути підходи до написання програм.

Відповідно до ООП, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю. З’являється поняття клас.

Також потрібно згадати про принципи ООП: успадкування – підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть реалізовувати свої власні. Успадкування може бути одиничне та множинне (використання множинного може бути обмежено мовою програмування); інкапсуляція - приховування деталей про роботу класів від об'єктів, що їх використовують або

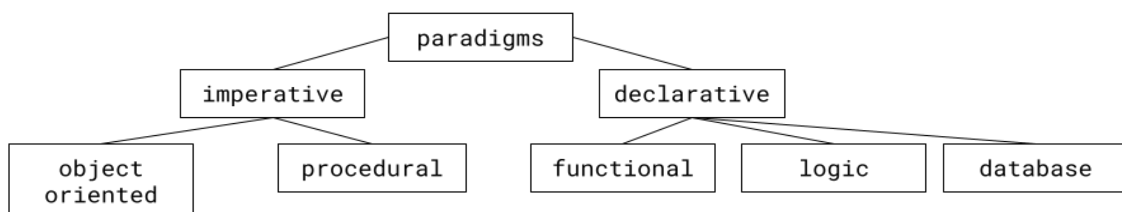


Рис. 1. Види ППр

надсилають їм повідомлення; абстрагування – спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми; поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Також ще є агрегація, композиція та асоціація. ООП зручне у використанні за рахунок опису ПЗ в шаблонах людського мислення. Тобто розробник може: дивлячись на певний реальний об'єкт зробити його програмну копію; використовуючи інкапсуляцію помістити властивості та дії, що може виконувати об'єкт поточного класу; вибрати використання потрібного рівня абстракції. Тобто для платіжної системи та для онлайн гри потрібні різні характеристики умовного об'єкта класу User. ООП також простіше для сприйняття менеджментом та замовником за рахунок опису класів та об'єктів. Композиція, агрегація, асоціація - чітко описують взаємодії між об'єктами (будування одних на основі інших). Наслідування описує механізми доповнення існуючих класів новим функціоналом. Проте, на жаль, на даному етапі це все ще доволі абстрактний підхід. Адже, наприклад, для реалізації ієрархії User в певній системі онлайн оплати може бути велика кількість підходів. Також впливає те, що в деяких варіантах відразу враховується предметна область, тобто розробник приймає рішення під впливом бізнес-логіки. Хоч професію розробника можна вважати, частково, креативною, але в цьому не завжди є необхідність, іноді навіть це проблема для командної роботи. Якщо розглянути веб-розробку, то для неї існують певні шаблони.

## 2. Приклад побудови проєкту node.js

Розглянемо реалізацію певного Web-API за допомогою nodejs. Перед початком варто звернути увагу на особливості роботи з nodejs: асинхронність; специфічний підхід до ООП; динамічна типізація. Загалом node.js API в більшості має наступні складові частини: middleware – посередники, що опрацьовують вхідні запити та можуть його мутувати (використовується для авторизації, іноді – для валідації); router – маршрутизатор запитів; controller – обробник запитів, що делегує обробку на сервіси; service – модуль, що виконує бізнес-логіку. Тобто запит проходить наступний шлях (рис. 2)

Поточний підхід доволі простий у виконанні, проте має низку проблем. Даний підхід передбачає під собою велику кількість middlewares, іноді це призводить до того, що бізнес-логіка проникає в них. Також controller і service зростаються з розмірах та мають високу прив'язку один-до-одного. Тобто їх неможливо буде в майбутньому замінити. Service виконує значну частину дій, таким чином змішуються бізнес-логіка, інфраструктурний код та код фреймворку. Також, часто об'єкт request проходить глибоко в бізнес-логіку, що призводить до високого рівня залежності.

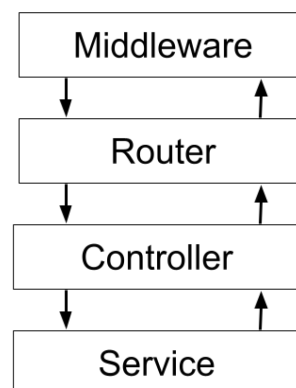


Рис. 2. Складові частини node.js API

Розглянемо популярний спосіб обробки запиту. Веб-додаток має авторизовану API, тобто тільки авторизовані користувачі можуть використовувати поточний додаток. Файл (рис. 3) відповідає за роботу з бізнес-логікою, а саме – створення завдань.

Приклад побудови маршрутизації для роботи із завданнями містить:

– read_a_task – обробник запиту на отримання завдання по ідентифікатору.

– create_a_task – обробник запиту на створення завдання.

Так виглядає обробник створення завдання та повернення завдання за ідентифікатором. На перший погляд код мініатюрний. Зручно написаний знаходиться в одному файлі відразу видно, що відбувається в кожному обробнику. Зараз обробники мають мінімальну кількість коду, проте такий код краще не використовувати в реальних проектах, в деяких випадках, навіть в тестовій розробці так ліпше не писати. В реальних умовах файл в собі може містить методи, що взаємодіють з базою даних (БД), файловою системою, використовують сторонні сервіси. Також потрібно враховувати процес валідації та формування відповіді для клієнта.

Далі зображено більш реалістичний код (рис. 4). Він виконує валідації, завантаження файлів, збереження даних в БД та роботу із зовнішнім сервісом. Відразу помітно проблему перетину інфраструктурного рівня та рівня бізнес-логіки. Файлі містить код маршрутизації та обробки.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

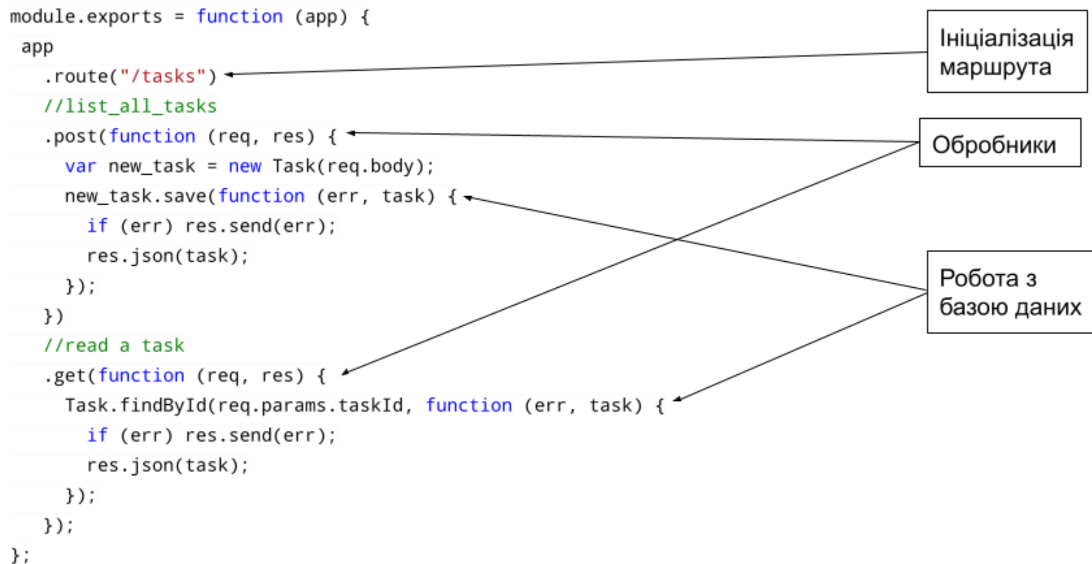


Рис. 3. Загальний приклад роботи за маршрутами

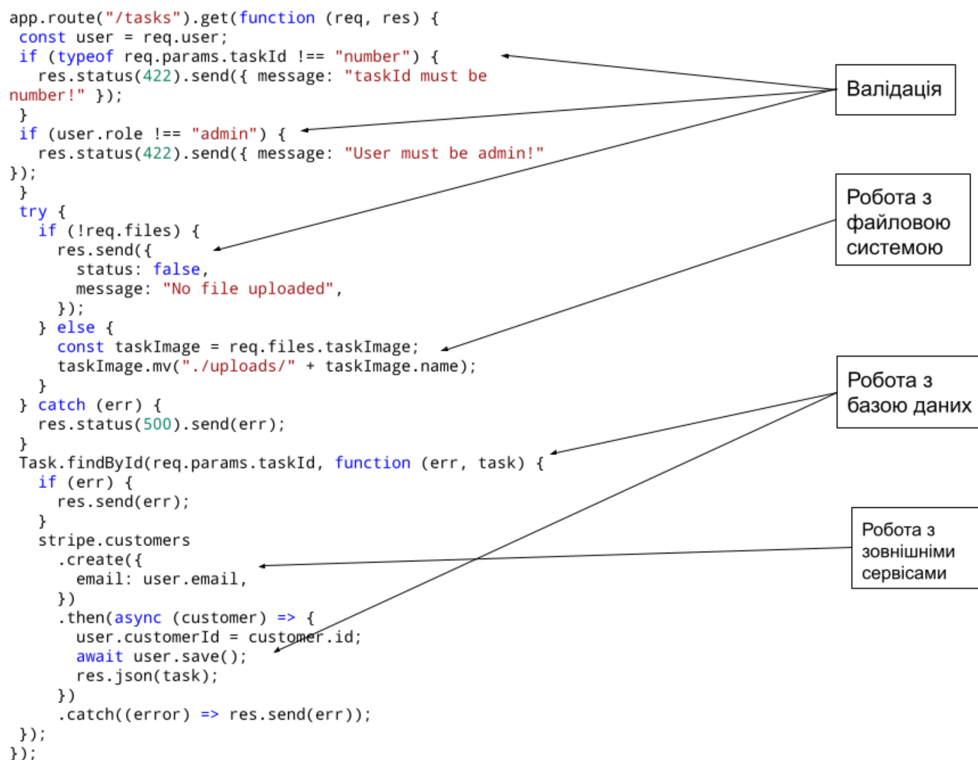


Рис. 4 Приклад реальної побудови маршрутизації для роботи із завданнями

Метод read_a_task має ‘велику відповідальність’. Загалом даний файл порушує перший принцип «SOLID» [6]. Метод виконує прості дії, проте навіть на даному етапі його вже складно рефакторити та додавати нові зміни. Варто врахувати, що іноді є необхідність валідувати 5-10 полів не тільки на тип, а і на діапазон значень, відповідність до конфігурацій.

Наступна проблема при створенні додатка за допомогою node.js – це надлишкова варіативність. Якщо оглянути метод read_a_task помітно, що при валідації полей використано один підхід повернення помилки, при валідації зображення формат помилки зовсім інший. Це призводить до складності обробки помилок, що може призвести до краху ПЗ. Використання асинхронного та синхронного коду також вносить певну долю варіативності. Це

призводить до використання конструкції try..catch разом з promise catch. Використання callback function, async/await, Promise – це все методи роботи з асинхронним кодом. Якщо розглядати дану проблему на рівні різних проектів, то така варіативність призводить до того, що на кожному проекті розробник повинен не тільки розібратися із предметною областю, а із підходом до написання коду. Іноді існують ситуації коли розробник вимушений дотримуватися стиля та логіки, яка різна в кожному файлі в межах одного проекту.

## 3. Формалізація моделі вибору шаблону проектування

Якщо із кожної дії відкинути бізнес проблеми, описати завдання професійними термінами, можна отримати загальні визначення для різних дій. Наприклад є два бізнес завдання: потрібно створити код, який буде створювати користувача з ім'ям та прізвищем, та вносити його до БД; потрібно створити код, який буде кожну годину записувати стан сервера в БД. Якщо абстрагуватися від дій, що призводять до виконання даного коду, то завдання можна описати так: потрібний код, що створює модель певного типу та у випадку успіху дана модель записується в БД у відповідному форматі. Список полів, назва таблиці - це додаткові характеристики від яких також можна абстрагуватися. Отже отримано загальний патерн для створення моделей. Є можливість таким чином описувати велику кількість задач.

Саме цей підхід описано у роботі [7]. Патерни (шаблони) проектування (ШП) – це напрацьовані ефективні підходи, техніки та правила вирішення задач при створенні ПЗ. ШП не прив’язуються до певної мови програмування і можуть бути застосованими в основному незалежно від конкретної мови. На відміну від написаного коду, ШП не можна просто взяти й впровадити в програму. ШП надають певні правила для взаємодії між класами, об’єктами, але вони абстраговані від бізнес-логіки. Можна сказати, що це паралельний рівень абстракції.

ШП дають можливість систематизувати процес розробки та дозволяють інженерам використовувати абстрактні одиниці взаємодії. Найбільш ефективним є використання ШП на етапі проектування, тому що: зменшується час на прийняття рішень та обговорення (за рахунок використання більш абстрактного опису класів); використовується стандартизований підхід до розроблення (кожний член команди використовує загальні правила розробки, що ідентичні для різних проектів); при зміні робочого ресурсу адаптація персоналу не потрібна (можливо розробляти уніфіковані класи, що в подальшому можуть використовуватися під час реалізації інших проектів). Кожен ШП описує певну кількість дій та характеристик, накладає правила та обмеження, проте залишається досить абстрактним. Таким чином класифікація ШП (табл. 1) за їх призначенням виглядає наступним чином: породжуючі – надають рекомендації та техніки для створення нових об’єктів; поведінкові – надають рекомендації для реалізації тої чи іншої поведінки-функції існуючого об’єкта; структурні – розглядають питання взаємодії між собою існуючих об’єктів. Загалом існує велика кількість ШП, кожна команда має можливість створювати свої внутрішні реалізації, вигадувати нові ШП, але в даній роботі за основу використано ШП з роботи [8] через простоту структури та поширеність.

Таблиця 1

Список патернів

| **Рівень\ Мета** | **Пород-жуючі** | **Струк-турні** | **Поведінки** |
|---|---|---|---|
| **Клас** | Factory Method | Adapter | Template Method |
| **Об’єкт** | Abstract Factory; Singleton; Prototype; Builder | Adapter; Decorator; Proxy; Composite; Bridge; Flyweight; Facade | Iterator; Command; Observer; Visitor; Mediator; State; Strategy; Memento; Chain Of Responsibility |

ШП, як правило побудовані з концептів, які є базовими елементами діаграми класів мови UML. В ході побудови моделі проєкту на основі метамоделі мови UML була запропонована наступна нотація :

$$Oprj = \langle Cprj, Rprj, Fprj \rangle,$$

де $Cprj = \{c_1^{prj}, \ldots, c_i^{prj}\}$ – множина концептів побудованих на основі елементів мови UML: "Class", "Object", "Interface", "Relationship" та інші; $Rprj$ – множина зв'язків між концептами, що описують відносини між елементами мови UML; $Fprj$ – множина функцій інтерпретації, визначених на множині $Rprj$.

Формально уявлення ШП можна представити таким чином:

$$\mathrm{O}_{\mathrm{tmp_i}}^{\mathrm{prj}} = \{\mathrm{inst}(\mathrm{C}_1^{\mathrm{prj}}), \ldots, \mathrm{inst}(\mathrm{C}_{\mathrm{rel1}}^{\mathrm{prj}}), \ldots \mathrm{r}_{\mathrm{sameAs}}\},$$

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

де $\text{inst}(C_1^{\text{prj}})$ - екземпляр концепту, побудованої на основі мета-схеми мови UML; $\text{inst}(C_{\text{rel1}}^{\text{prj}})$ - відношення між елементами ШП, представлене у вигляді примірника концепту; $rsameAs$ - еквівалентність примірників, що входять до нього.

Для виявлення міри вираженості ШП у проєкті ПЗ пропонується наступний коефіцієнт:

$$\mu_{prj,tmp} = \frac{|C_{prj} \cap C_{tmp}| + |R_{prj} \cap R_{tmp}|}{|C_{tmp}| + |R_{tmp}|},$$

де $Cprj$ і $Ctmp$ - екземпляри концепту проєкту ПЗ або ШП, $Rprj$ і $Rtmp$ - екземпляри відносин проєкту ПЗ або ШП.

Розглянемо пропонований коефіцієнт для проєкту node.js

## 4. Приклад використання ШП у проектах node.js

Розглянемо ідею сумісного використання ШП у проектах node.js (рис. 5). Існують практичні рекомендації з використання цього підходу (наприклад, реалізація підключення до БД за допомогою ШП singleton) але відсутність моделі та методики її використання не дає змогу їх широкого розповсюдження.

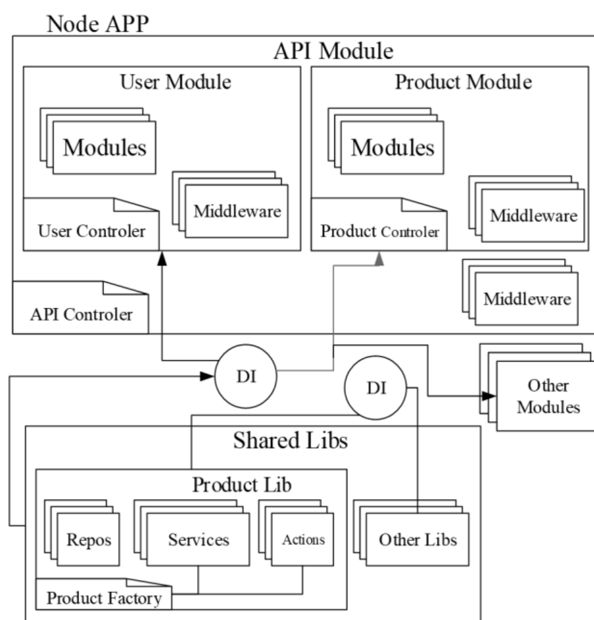


Рис. 5. Схема взаємодії елементів

Розглянемо опис елементів проекту node.js:

– **module** – частина додатку, потрібна для розбиття додатку на частини за відповідальністю, наприклад user module, product module.

– **validator** – перевірка відповідності вхідних даних;

– **controller** – обробник запитів, що об'єднує в собі router та обробники, делегує обробку на сервіси або action;

– **service** – модуль, виконує бізнес-логіку, може використовувати action, repository, service;

– **dto** – data transfer object. Запобігає зануренню request в бізнес-логіку. Використовується для передачі даних між компонентами;

– **middleware** – посередник, що опрацьовує вхідні запити може його мутувати. Використовується для авторизації та аутентифікації;

– **action** – уніфікована бізнес-дія, модуль, що інкапсулює складну бізнес-логіку;

– **adapter** – модуль для трансформації даних, для відправлення даних по мережі;

– **factory** – модуль для об'єднання різних компонентів, для реалізації dependency injection(DI);

– **repository** – сервіс для роботи з БД.

Це загальний список компонентів, що запропоновано використовувати для побудови ПЗ node.js. Далі буде приведено список елементів та ШП за допомогою яких ці елементи реалізовані.

Головним елементом є module. Кожен module в собі обов'язково має містити controller та може містити інші module. Умовно даний підхід схожий на побудову дерева, де кожен module як гілка може мати інші гілки (module) або бути листком (controller). Це реалізація ШП Composite (рис.6). Composite - це структурний ШП, що дає змогу згрупувати декілька об'єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об'єкт.

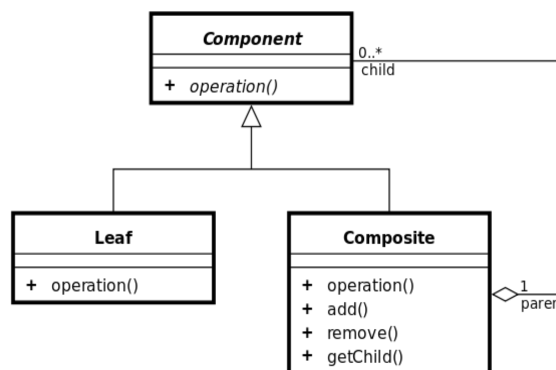


Рис. 6. Схема ШП Composite [6]

Кожен module та controller це Component. Component – це інтерфейс який реалізуються Leaf (controller) та Composite (module).

Нажаль в JavaScript не має можливості скористатися interface тому це буде не реалізація, а

наслідування, тобто Component - батьківський клас ($\mu prj_{\text{node.js}}, tmp_{\text{Composite}}$ =0,8).

Наступний елемент це validator (в поточному підході це middleware), що перевіряє вхідний запит на відповідність до бажаних даних. Наприклад, є маршрут за допомогою якого є можливість створити товар. При створенні товару необхідно переконатися, що поле title буде string та матиме довжину більше 6 символів. Саме для цього потрібен елемент validator. Отже це обробник, що вирішує чи можливо обробити даний запит (в іншому випадку повідомляє клієнту, що запит не відповідає вимогам). Також middleware використовується для перевірки доступу до ресурсів, авторизації та автентифікації. Отже фактично можна вважати, що це реалізація ШП Chain of responsibility або ШП Ланцюжок обов'язків (рис. 7).

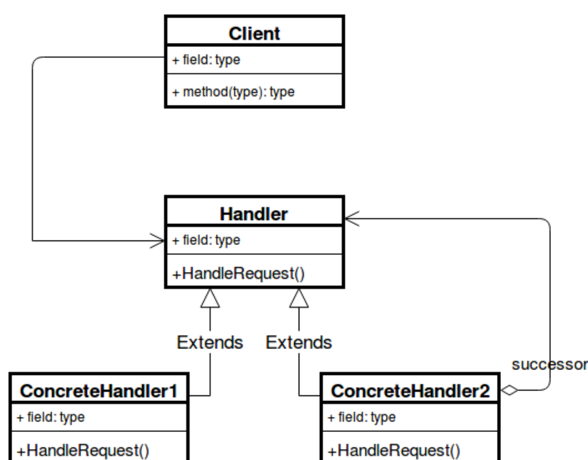


Рис. 7. Схема ШП Chain of responsibility [6]

Ланцюжок обов'язків – це поведінковий ШП проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком або припинити виконання запиту та повідомити про помилку. Підхід даного ШП досить прозорий кожен обробник виконує дію або перевірку та передає запит іншому обробнику або зупиняє роботу ланцюжка. Це дуже популярний підхід в node.js. Кінцевий обробник запиту також можна вважати middleware, тобто останнім Handler. За вірного використання ШП досяжним є якісний рівень відповідальності кожного обробника (SOLID). Було розраховано коефіцієнт міри вираженості ШП Chain of responsibility у проєкті node.js (0,7).

Вище розглянуто основні елементи ПЗ, які орієнтовані на інфраструктуру та роботу із зовнішнім клієнтом. Проте не розглянуто класи бізнес-логіки, які є також важливою частиною ПЗ, та також реалізовані повністю із використанням ШП.

## Висновки

Вибір теми дослідження був пов'язаний з актуальністю використання ШП [9, 10]. ШП дозволяють не прив'язуються до певної мови програмування і можуть бути застосованими незалежно. ШП надають певні правила для взаємодії між класами, об'єктами, але вони абстраговані від бізнес-логіки. Можна сказати, що це паралельний рівень абстракції. ШП дають можливість систематизувати процес розробки та дозволяють інженерам використовувати абстрактні одиниці взаємодії.

В даній роботі розглянуто сучасні методології та парадигми проектування, сформовано класифікацію у вигляді деревоподібної структури за поділом на декларативні та імперативні підвиди. У рамках цього дослідження використано об'єктно-орієнтовану методологію, як найпоширенішу парадигму проектування. Розглянуто приклад побудови інформаційної системи проєкту node.js. Проаналізовано основні помилки, що виникають при розробці та написанні коду роботи із зовнішнім клієнтом. Розглянуто елементи проєкту node.js та концепцію структуризації їх взаємозв'язку з існуючими шаблонами проектування. Розглянуто приклад практичної реалізації проєкту node.js та його зв'язок з шаблонами проектування Composite, та Chain of responsibility. Запропоновано та розраховано коефіцієнт міри вираженості ШП у проекті node.js.

Розглянута концепція є першим етапом розробки методу пошуку та відокремлення ключових частин проекту ПЗ та реалізації кожної частини ПЗ за допомогою відповідного ШП, що буде обрано відповідно до його характеристик.

## References (GOST 7.1:2006)

*1. Comparison of Metoheuristic Search Methods for the Task of Choosing a Rational Set of Measures to Risks' Respond [Text] / O. K. Pohudina, A. D. Morikova, B. I. Haidabrus, S. I. Kiyko, E. A. Druzhinin // Conference on Integrated Computer Technologies in Mechanical Engineering–Synergetic Engineering, Cham, 25 October 2020. – 2020. – Vol. 188. – P. 657-666.*

*2. Kisling, E. Transitioning from Waterfall to Agile: Shifting Student Thinking and Doing from Milestones to Sprints [Text] / E. Kisling // Proceedings of Southern Association for Information Systems. – 2019. – Vol.14. – P. 1-2.*

*3. A parallel implementation strategy for meshless methods based on the functional programming paradigm [Text] / M. Barbosa, J. C. Faria Telles, J. A. Santiago, E. F. Junior, E. G. Araújo Costa // Advances in Engineering Software. – 2021. – Vol. 151. – Article Id: 102926.*

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

*4. Risk assessment across life cycle phases for small and medium software projects. [Text] / M. Bilal, A. Gani, M. Liaqat, N. A. Bashir, N. A. Malik // Journal of Engineering Science and Technology. – 2020. – Vol. 15, no. 1. – P. 572-588.*

*5. Евланов, М. В. Модели паттернов проектирования требований к информационной системе на уровне данных [Текст] / М. В. Евланов // Радіоелектронні і комп'ютерні системи. – 2014. – № 1(65). – С. 128–138.*

*6. Gamma, E. Design patterns: elements of reusable object-oriented software [Text] / E. Gamma. – India : Pearson Education, 1995. – 395 p.*

*7. Modularizing design patterns with aspects: a quantitative study [Text] / A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. Staa // Transactions on Aspect-Oriented Software Development. – Berlin, 2006. – Vol. 3880, no.18. – P. 36-74.*

*8. Ortin, F. Design and evaluation of an alternative programming paradigms course [Text] / F. Ortin, J. M. Redondo, J. Quiroga // Telematics and Informatics. – 2017. – Vol. 34, iss. 1. – P. 813-823.*

*9. Hanam, Q. Discovering bug patterns in JavaScript [Text] / Q. Hanam, F. S. de M. Brito, A. Mesbah // Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – 2016. – P. 144-156.*

*10. An UML profile for representing real-time design patterns [Text] / H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, B. Sadeg // Journal of King Saud University - Computer and Information Sciences. – 2018. – Vol. 30, iss. 4. – P. 478-497.*

## References (BSI)

1. Pohudina, O. K., Morikova, A. D., Haidabrus, B. I., Kiyko, S.I., Druzhinin, E. A. Comparison of Metoheuristic Search Methods for the Task of Choosing a Rational Set of Measures to Risks' Respond. *Conference on Integrated Computer Technologies in Mechanical Engineering–Synergetic Engineering, Cham, 25 October 2020,* Cham, Springer, 2020, vol. 188, pp. 657-666.

2. Kisling, E. Transitioning from Waterfall to Agile: Shifting Student Thinking and Doing from Milestones to Sprints. *Proceedings of Southern Association for Information Systems*, 2019, vol. 14, pp. 1-2.

3. Barbosa, M., Faria Telles, J. C., Santiago, J. A., Junior, E. F., Araújo Costa, E. G. A parallel implementation strategy for meshless methods based on the functional programming paradigm. *Advances in Engineering Software*, 2021, vol. 151, article id: 102926.

4. Bilal, M., Gani, A., Liaqat, M., Bashir, N.A., Malik, N. A. Risk assessment across life cycle phases for small and medium software projects. *Journal of Engineering Science and Technology*, 2020, vol. 15, no. 1, pp. 572-588.

5. Yevlanov, M. V. *Modeli patternov proyektirovaniya trebovaniy k informatsionnoy sisteme na urovne dannykh* [Models of design patterns for information system requirements at the data level]. *Radioelektronni i komp'uterni sistemi – Radioelectronic and computer systems*, 2014, no. 1(65), pp. 128-138.

6. Gamma, E. *Design patterns: elements of reusable object-oriented software.* India, Pearson Education Publ., 1995. 395 p.

7. Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., Staa, A. Modularizing design patterns with aspects: a quantitative study. *Transactions on Aspect-Oriented Software Development,* Berlin, Springer, 2006, vol. 3880, no. 18, pp. 36-74.

8. Ortin, F., Redondo, J. M., Quiroga, J. Design and evaluation of an alternative programming paradigms course. *Telematics and Informatics*, 2017, vol. 34, iss. 1, pp. 812-823.

9. Hanam, Q., Brito, F. S. de M. Discovering bug patterns in JavaScript. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 144-156.

10. Marouane, H., Duvallet, C., Makni, A., Bouaziz, R., Sadeg, B. An UML profile for representing real-time design patterns. *Journal of King Saud University - Computer and Information Sciences*, 2018, vol. 30, iss. 4, pp. 478-497.

*Надійшла до редакції 5.01.2021, розглянуто на редколегії 16.02.2021*

**EVALUATION OF USE OF DESIGN TEMPLATES IN THE SOFTWARE DEVELOPMENT**

***M. Bychok, O. Pohudina***

The **subject** of study in the article is software development processes using design patterns. The **aim** is to improve the quality of modern software development projects through the use of experience and knowledge, to build software subsystems that are focused on infrastructure and work with an external client. **Objectives**: to review the methodology, programming paradigms and the possibility of their application at the design and coding stages of the software development life cycle; development of the concept of using design patterns in software design as knowledge available for reuse, propose an approach to the practical implementation of design patterns to node.js projects. The **models** used are the Composite design pattern, the Chain of responsibility design pattern. The used **methodologies** are object-oriented programming, as the most common programming paradigm, a unified modeling language UML for displaying the structure of design patterns. The following **results** are obtained. Modern methodologies and design paradigms are considered, a classification is formed in the form of a tree structure with a division into declarative and

imperative subspecies, it is concluded that within the framework of the study we will use an object-oriented methodology as the most common design paradigm. An example of building an information system of the node.js project is considered. Analyzed the main errors that arise when developing and writing code for working with an external client. The elements of the node.js project and the concepts of structuring their relationship with existing design patterns are considered. An example of a practical implementation of a node.js project and its relationship with the Composite and Chain of responsibility design patterns is considered. In this connection, the work provides the structure of these templates. **Findings.** The scientific novelty of the results obtained is as follows: the model of design patterns was further developed through their use in the concept of building a node.js application, which makes it possible to improve the quality of interaction between the project team and reduce its execution time.

**Keywords:** software; programming paradigm; design patterns; node.js project.

## ОЦЕНКА ИСПОЛЬЗОВАНИЯ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

***М. А. Бычок, О. К. Погудина***

**Предметом** изучения в статье является процессы разработки программного обеспечения (ПО) с использованием паттернов проектирования. **Целью** является повышение качества проектов разработки современного ПО за счет использования опыта и знаний, по построению подсистем ПО, которые ориентированы на инфраструктуру и работу с внешним клиентом. **Задачи:** выполнить обзор методологий, парадигм программирования и возможности их применения на этапах проектирования и кодирования жизненного цикла разработки ПО; разработка концепции применения паттернов проектирования при проектировании ПО как знаний, доступных для повторного использования, предложить подход к практической реализации паттернов проектирования до проектов node.js. Используемыми **моделями** являются модель шаблона проектирования Composite, модель шаблона проектирования Chain of responsibility. Используемыми **методологиями** являются объектно-ориентированное программирование, как самая распространенная парадигма программирования, унифицированный язык моделирования UML для отображения структуры шаблонов проектирования Получены такие **результаты**. Рассмотрены современные методологии и парадигмы проектирования, сформирована классификация в виде древовидной структуры с разделением на декларативные и императивные подвиды, сделан вывод, что в рамках исследования будем использовать объектно-ориентированную методологию, как самую распространенную парадигму проектирования. Рассмотрен пример построения информационной системы проекта node.js. Проанализированы основные ошибки, возникающие при разработке и написании кода работы с внешним клиентом. Рассмотрены элементы проекта node.js и концепции структуризации их взаимосвязи с существующими шаблонами проектирования. Рассмотрен пример практической реализации проекта node.js и его связь с шаблонами проектирования Composite, и Chain of responsibility. В связи с чем в работе предоставлено структура этих шаблонов. **Выводы**. Научная новизна полученных результатов заключается в следующем: получили дальнейшее развитие модели шаблонов проектирования за счет их использования в концепции построения приложения node.js, что дает возможность повысить качество взаимодействия команды проекта, сократить его сроки исполнения.

**Ключевые слова:** программное обеспечение; парадигма программирования; шаблоны проектирования; проект node.js.

**Бичок Максим Олександрович –** аспірант кафедри інформаційних технологій проектування, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

**Погудіна Ольга Костянтинівна –** канд. техн. наук, доцент кафедри інформаційних технологій проектування, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

**Maksym Bychok** – PhD student in Department "Information Technology of Design", National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: bychokmaks@gmail.com, ORCID: 0000-0002-2030-5579.

**Olha Pohudina** – PhD in Technical, Associate Professor in Department "Information Technology of Design", National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: ok.pogudina@gmail.com, ORCID: 0000-0001-5689-2552, Scopus Author ID: 57204907264,
ResearcherID: Y-1277-2019, https://scholar.google.com.ua/citations?user=-yKGgW8AAAAJ&hl=ru.