DOI: 10.20535/1970.67(1).2024.306737
УДК 004.4

ПРОЄКТУВАННЯ АРХІТЕКТУРИ АВТОМАТИЗОВАНОЇ СИСТЕМИ СТВОРЕННЯ СУПРОВІДНОЇ ДОКУМЕНТАЦІЇ ОСВІТНЬОГО ПРОЦЕСУ

Цибульник С. О., Накорик В. В., Півторак Д. О.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
E-mail: tsybulnik.s.a@gmail.com, p_diana@i.ua

Архітектура програмної автоматизованої системи є основним джерелом якості програмних та програмно-апаратних систем. Вплив архітектури полягає в тому, що вона визначає, наскільки швидко та ефективно розробник здатний проаналізувати, зрозуміти, перевірити, розширити та підтримувати програмну автоматизовану систему. Зміни в архітектурі програмної системи мають високу вартість через її складність та можливість руйнування під час розширення.

У даний час більшість знань та інформація про дизайнерські рішення, на яких базується архітектура, неявно вбудовані в неї, що призводить до виникнення проблем під час процесу розроблення, зокрема при програмній реалізації її структурних елементів.

Визначено, що, незважаючи на довгий шлях, який пройшла еволюція архітектури програмних систем, на сьогодні існує дуже мало об'єктивних, повторюваних та емпірично обґрунтованих методологій та інструментів для проєктування та аналізу архітектури. Здебільшого архітекторами програмного забезпечення є програмісти з великим досвідом практики розроблення програмного забезпечення. Враховуючи цей досвід, вони чітко розуміють, що помилки в проєкті архітектури є причинами проблем нижчого рівня, що проявляються в програмному коді. Найчастіше програміст відчуває, коли архітектура його проєкту має низьку якість, тому що існує велика кількість технічних недоліків, кількість яких з часом лише зростає. Але більшість проєктів продовжують виконання, що призводить до зниження якості готової програмної автоматизованої системи.

Саме тому метою даної роботи є проєктування та документування програмної архітектури на прикладі автоматизованої системи створення супровідної документації освітнього процесу, щоб зробити перший крок у напрямку розуміння взаємозв'язків та впливу прийнятих проєктних рішень на кодову базу.

Для досягнення мети було обрано клас багаторівневих архітектур, серед яких найбільш поширеною є трирівнева. Розглянуто особливості реалізації відкритої та закритої трирівневих архітектур. На основі архітектурного шаблону MVC розроблено проєкт архітектури автоматизованої системи створення супровідної документації освітнього процесу. Задокументовано основні підсистеми та елементи даних.

Ключові слова: *автоматизована система; архітектура; програмне забезпечення; MVC; документування.*

Вступ

Архітектура програмної автоматизованої системи є основним джерелом якості та (з точки зору довговічності) стійкості програмних та програмно-апаратних систем. Це пов'язано з тим, що саме архітектура впливає на те, наскільки швидко та правильно розробник здатний проаналізувати, зрозуміти, перевірити, розширити та підтримувати програмну автоматизовану систему [1]. Внесення змін до архітектури програмної системи має високу вартість через те, що вона є дуже складною та може руйнуватися під час розширення. Наразі майже всі знання та інформація про проєктні рішення, на яких базується архітектура, неявно вбудовані в неї, що призводить до виникнення проблем під час процесу розроблення, тобто при програмній реалізації її структурних елементів.

Сьогодні зазвичай під архітектурою програмної автоматизованої системи прийнято вважати структурно-функціональну модель, яка описує структуру (склад) системи (підсистеми, рівні, компоненти тощо), функції елементів, їх взаємозв'язки та взаємодію (інтерфейси), а також правила їх композиції. Проте дане визначення не єдине. Якщо дивитися з огляду на структури, то архітектура програмної автоматизованої системи охоплює сукупність компонентів (структур), які складаються з елементів програмного забезпечення, їхніх взаємозв'язків та властивостей [2, 3]. З іншого боку [4] архітектуру можна вважати рядом проєктних рішень, які забезпечують якість та стійкість програмної системи в процесі її розроблення, розширення та супроводу.

Отже, під архітектурою програмної автоматизованої системи варто розуміти не будь-яку структурно-функціональну модель, а лише таку, в якій внутрішні системні інтерфейси мають точний опис, який виключає двояке розуміння функціональних особливостей, а також забезпечуються якість та стійкість системи в процесі розроблення та супроводу.

За останні п'ятдесят років у сфері проєкту-

вання архітектури програмних автоматизованих систем було створено багато важливих та інноваційних методологій та засобів, які їх реалізовують [5]. Особливий вплив на розвиток даного напрямку здійснили принципи об'єктно-орієнтовного проєктування. Ідеї абстракції та інкапсуляції (приховування деталей реалізації) змінили погляд програмістів, їх думки та бачення програмних автоматизованих систем завдяки розробленню новітніх принципів програмування (наприклад, SOLID), архітектурних шаблонів [6-8], фреймворків тощо.

Однак, незважаючи на довгий шлях, який пройшла еволюція архітектури програмних систем, на сьогодні існує дуже мало об'єктивних, повторюваних та емпірично обгрунтованих методологій та інструментів для проєктування та аналізу архітектури. Зазвичай архітекторами програмного забезпечення стають програмісти, які мають декілька десятиліть стажу практики розроблення програмного забезпечення [5]. Враховуючи їх досвід, вони чітко розуміють, що помилки в проєкті архітектури є корінними причинами проблем нижчого рівня, що проявляються в програмному коді [6, 9]. Найчастіше розробник (програміст) знає або, принаймні, відчуває, коли архітектура його проекту має низьку якість, тобто існує величезна і постійно зростаюча кількість технічних недоліків, через які продуктивність програмної системи падає. Але більшість розробників через брак досвіду і знань про програмну архітектуру мають проблеми з визначенням того, як і чому це відбувається, і ще менше розуміють, як це виправити. Тому їм майже неможливо довести до керівництва, що архітектура проєкту та кодова база мають бути ретельно проаналізовані та перероблені [5]. Оскільки розробники не мають необхідних даних та знань, вони не не можуть надати переконливих доказів того, що у разі проведення рефакторингу ситуація покращиться. З цієї причини більшість проєктів продовжують виконання, що призводить до зниження якості готової програмної автоматизованої системи.

Для багатьох галузей промисловості та суспільства [10-13] відсутня необхідність в обговоренні великої важливості використання програмних автоматизованих систем достатнього рівня якості (наприклад, не крихких [14]).

Програмні системи стали надзвичайно поширеними, а разом з ними виросла залежність суспільства від автоматизованих систем, які керуються програмним забезпеченням. Тому розроблення інноваційних і ефективних способів проєктування якісної архітектури для автоматизованих систем є реальною потребою сьогодення. Не менш важливим є знання того, як оцінити фактичний рівень стійкості та надійності будь-якого програмного продукту. Щоб цього досягти, необхідно розробити чіткі методолігії архітектурного проєктування [15], які чітко та об'єктивно систематизують необхідний перелік дій для реалізації кожної з існуючих груп архітектур.

Постановка задачі

Важливою особливістю архітектури є те, що всі підсистеми, які виділяються при декомпозиції загальної задачі, мають цілком конкретне функціональне призначення. Понад те, уточнення функцій кожної підсистеми досягається за допомогою опису її інтерфейсів, тобто повного набору взаємозв'язків (алгоритмічних, інформаційних тощо) з іншими підсистемами. Фактично, опис інтерфейсів підсистем визначає їх зовнішню поведінку, тоді як внутрішні механізми реалізації цієї поведінки в межах архітектури не розглядаються і залишаються на розсуд розробників, які будуть реалізовувати певну програмну архітектуру. Якість реалізації напряму залежить від якості проєктування та документування архітектурних рішень.

Неправильне формування вимог до розроблюваної автоматизованої програмної системи призводить до помилок в декомпозиції з наступним їх документуванням, що викликає порушення логіки взаємозв'язків та збоїв у взаємодії компонентів між собою. Детальний аналіз прийнятих архітектурних рішень під час декомпозиції системи дозволяє оцінити коректність постановки задачі та підвищити точність документування для забезпечення достатнього рівня якості в подальшому процесі розроблення. Саме тому метою даної роботи є проєктування та документування програмної архітектури на прикладі автоматизованої системи створення супровідної документації освітнього процесу.

Відкрита та закрита багаторівневі архітектури

Найбільш популярним класом програмної архітектури сьогодні є багаторівнева. Її особливість полягає в тому, що структура системи поділяється на фізичні рівні, кожен з яких реалізовується певною апаратною частиною (наприклад, сервер, база даних тощо), та логічні шари, які реалізовуються у вигляді програмного забезпечення в межах відповідного рівня. Класична реалізація багаторівневої архітектури містить три рівні та чотири шари і називається трирівневою. Існує два способи її реалізації, а саме [16]: закрита та відкрита трирівневі архітектури.

Закрита трирівнева архітектура (рис. 1) будується на принципі інкапсуляції даних, коли кожен шар знає про існування лише сусідніх шарів у межах доступних інтерфейсів. Будь-яка взаємодія з усіма іншими шарами неможлива через відсутність інформації про їх існування.

На рис. 1 кожен шар позначено як «Закритий». Тобто для того, щоб ініційований користувачем запит пройшов весь шлях від шару графічного подання до шару сховища даних, необхідно послідовно відвідати кожен шар. Така реалізація трирі-

вневої архітектури гарантує виконання умови низького зчеплення (низька крихкість системи) між програмними сутностями, тобто внесені в один з шарів зміни ніяким чином не впливатимуть на інші шари окрім розширення їх інтерфейсів.

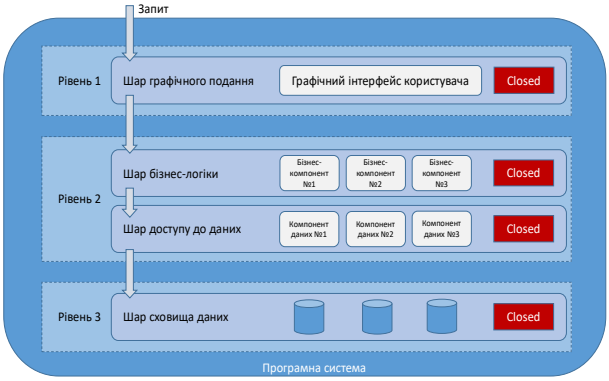

Рис. 1. Приклад закритої трирівневої архітектури

На відміну від закритої, відкрита трирівнева архітектура характеризується тим, що структурний поділ рівнів на окремі шари може відрізнятися від класичного (рис. 2). При цьому додаткові шари, які вводяться в структуру системи, є не обов'язковими, тобто не кожен запит користувача буде викликати наявні там функціональні можливості. Прикладом такого необов'язкового шару може бути шар сервісів [17-19]: сервіс викликається лише в тому випадку, коли користувач безпосередньо ініціює з ним взаємодію через елементи графічного інтерфейсу та пов'язану з ними бізнес-логіку. Прикладом сервісу може бути купівля додаткового контенту в багатокористувацьких іграх або купівля блокування рекламних пропозицій в деяких мобільних додатках.

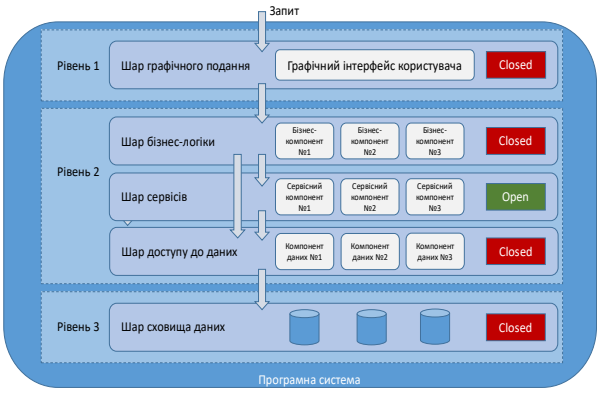

Рис. 2. Приклад відкритої трирівневої архітектури

На рис. 2 один з шарів позначено як «Відкритий». Таким чином, частина запитів від користувача буде ігнорувати даний шар та проходити від бізнес-логіки відразу до шару доступу до даних.

На відміну від попереднього випадку, подібне рішення може призводити до незначного підвищення швидкодії за рахунок переходу через один або декілька шарів під час обробки запиту, але може збільшувати крихкість системи через наявність більшої кількості взаємозв'язків між її елементами.

Отже, до переваг використання трирівневої архітектури можна віднести інкапсуляцію: логічні шари знають лише про існування сусідніх, взаємодія між ними відбувається через чітко визначені інтерфейси, а також жоден шар не знає нічого про внутрішню структуру суміжних шарів. Недоліком є лише збільшення мережевого трафіку через необхідність проходження запитом великої кількості шарів, у межах яких може навіть не відбуватися додаткова обробка даних.

Враховуючи все вищесказане, для проєкту автоматизованої системи створення супровідної документації освітнього процесу обрано закриту трирівневу архітектуру через її кращу відповідність принципам об'єктно-орієнтовного проєктування архітектури програмного забезпечення.

Проєкт архітектури програмної системи

Супровідна документація в освітньому процесі охоплює різноманітні документи, які встановлюють і фіксують особливості освітнього процесу, забезпечують правильну організацію та контроль над навчанням та оцінюванням здобувачів освіти. Весь супровідний документообіг у закладах освіти, як правило, створюється людиною вручну або за допомогою не дуже зручних інструментів, таких як блокнот, Word або Excel.

Створення автоматизованої системи дозволить привести всю супровідну документацію до єдиного прийнятого шаблону, централізувати перевірку та затвердження, а також дасть зручний інструмент для контролю версій.

Саме тому на рис. 3 запропоновано проєкт архітектури подібної автоматизованої системи, яка має класичну структуру: шар відображення, який відповідає за графічний інтерфейс користувача; шар бізнес-логіки, який містить основні елементи обробки та перетворення даних; шар доступу до даних, задача якого полягає в формуванні об'єктів або масиву даних, їх шифруванні та передачі в підсистему роботи з базою даних (БД); шар сховища даних, який містить підсистему роботи з БД та ряд БД.

Шар відображення містить лише одну підсистему, основне завдання якої полягає у наданні користувачу графічного інтерфейсу для зручної взаємодії з програмним забезпеченням системи.

Усього в дану підсистему входить три модуля, які відповідають за окремі графічні вікна з відповідним набором елементів, а саме:

1) *Модуль аутентифікації та реєстрації* – містить ряд полів для введення текстової інформації, яка дає змогу підтвердити наявність користувача в БД або створити нового, та кнопок підтвердження.

2) *Модуль відображення силабусів* – містить графічні елементи шаблону силабусу навчальної

дисципліни (поля для введення, випадаючі меню, списки тощо), які відповідають певному стандарту освіти, а також можливість перемикання між різними шаблонами. Взаємодія з елементами шаблону дає змогу користувачу сформувати силабус навчальної дисципліни.

3) *Модуль відображення індивідуальних планів* – містить графічні елементи шаблону індивідуального плану студента, які відображають інформацію про його поточний статус (ПІБ, факультет, курс, група, спеціальність, освітня програма), а також список нормативних та вибіркових дисциплін для вивчення в поточному навчальному році. Список навчальних дисциплін є інтерактивним, оскільки взаємодія з його елементами (назва, години, контрольні заходи, семестровий контроль, тощо) дозволяє побачити додаткову інформацію (у вигляді додаткового контекстного вікна) з відповідного силабусу, наприклад, перелік тем лекцій, практичних робіт, рейтингову систему оцінювання тощо.

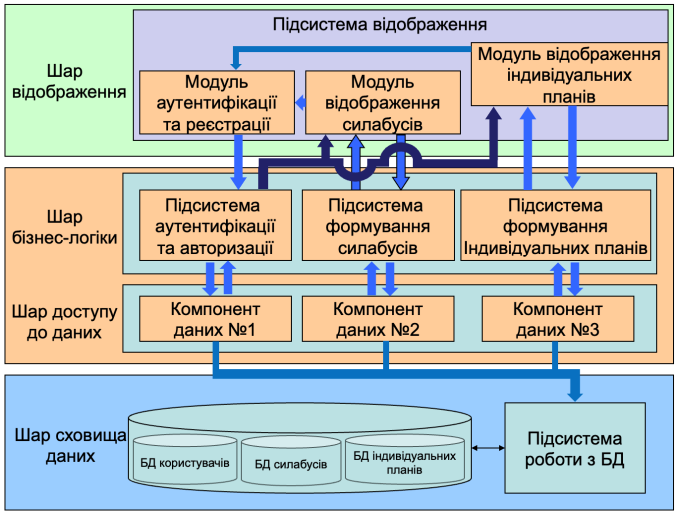

Рис. 3. Трирівнева архітектура автоматизованої системи створення супровідних документів освітнього процесу

Шар бізнес-логіки складається з трьох підсистем:

1) *Підсистема аутентифікації та авторизації.* Дана підсистема приймає на вхід масив даних, які ввів користувач під час роботи з модулем аутентифікації та реєстрації. Отримана інформація обробляється для визначення її типу (аутентифікація/реєстрація) та формується масив відповідних даних для подальшої передачі на наступний шар. Також дана підсистема обробляє інформацію, яка повернулася з БД, про результати аутентифікації, тобто підтвердження наявності даного користувача в системі.

Успішне проходження аутентифікації дає змогу користувачу отримати доступ до інших модулів залежно від поточної ролі користувача в БД (авторизація): до модуля відображення силабусів для ролі «викладач»; до модуля відображення індивідуальних планів для ролі «студент». Роль «адміністратор» має доступ до власного модуля відображення та підсистем, які на даному етапі не розглядаються в межах спроєктованої архітектури та даної роботи.

2) *Підсистема формування силабусів.* Дана підсистема приймає на вхід масив даних, які ввів користувач під час роботи з модулем відображення силабусів. Оскільки користувач системи постійно працює зі створеним за певним стандартом освіти шаблоном силабусу, весь набір даних відомий заздалегідь. Тому вхідний масив має статичну кількість стовпчиків, яка відповідає за елементи графічного вікна шаблону силабусу (поля для введення, списки тощо), та динамічну кількість рядків, яка відповідає за додаткові елементи, що створюються в межах передбачених груп (об'єкти, елементи списку тощо). Отриманий масив даних перевіряється на коректність та, у разі позитивного рішення, передається далі.

3) *Підсистема формування індивідуальних планів.* Дана підсистема слугує лише для відображення даних. Функціями, які вона виконує, є наступні: визначення типу отриманої з БД інформації (списки навчальних дисциплін, кількість годин/кредитів навчальної дисципліни, перелік тем лекцій тощо), перевірка масивів даних на коректність, обробка подій, викликаних натисканням на інтерактивні елементи в модулі відображення індивідуальних планів (наприклад, при натисканні на число лекційних годин з'являється додаткове контекстне вікно з переліком лекційних тем, які отримуються з БД і обробляються в даній підсистемі для подальшого відображення).

Шар доступу до даних складається з трьох компонентів даних, які виконують завдання шифрування отриманих з шару бізнес-логіки даних та їх передачу на наступний рівень, а також дешифрування отриманої з БД інформації.

Шар сховища даних складається з двох частин:

1) Бази даних. Даний елемент містить набір таблиць, які включають об'єднану певною логікою інформацію. Для коректної реалізації принципу інкапсуляції кожен вид інформації має свою БД для збереження, а саме: аутентифікація та авторизація містить дані входу та інформацію про роль, тобто права доступу, користувача; списки дисциплін та силабусів, які для них створено; шаблони силабусів; інформаційне наповнення силабусів; шаблони індивідуальних планів; інформаційне наповнення індивідуальних планів.

2) Підсистема роботи з БД. Якщо попередній рівень містив усі елементи бізнес-логіки, то на даному рівні знаходиться базова програмна логіка, яка відповідає за алгоритми пошуку та сортування, а також зміну даних (редагування, видалення, додавання).

У якості вхідних підсистема приймає зашифрований набір даних, які вона спочатку дешифрує. Наступним етапом обробки є визначення належності даних до певного класу, який указуватиме на БД, з якими потрібно вести взаємодію. У загальному може формуватися чотири типи запитів: зчитування, додавання, видалення та редагування інформації. Залежно від типу запиту будуть задіяні різні програмні модулі, наприклад, модуль пошуку працюватиме при виконанні всіх чотирьох запитів, а модуль сортування тільки для операції додавання.

Загальний алгоритм підсистеми роботи з БД виглядатиме наступним чином: дешифрувати вхідний масив даних; визначити якому компоненту належать ці дані; визначити тип запиту; отримати копію необхідної БД; провести визначені в запиті операції з даними (пошук, сортування, додавання тощо); сформувати відповідь на отриманий запит (наприклад, [true, "викладач"] для підсистеми аутентифікації та авторизації); зашифрувати відповідь; направити відповідь до необхідного компоненту даних.

На даному етапі розроблення проєкту архітектури програмної системи процес декомпозиції не зупиняється.

У майбутньому треба провести детальне проєктування, яке дозволить показати логіку взаємодії модулів в кожній підсистемі. Документування цієї логіки в сукупності з результатами документування системи на більш високому рівні, які отримані в даній роботі, надасть змогу отримати завершений проєкт системної архітектури.

Клас багаторівневої архітектури має багато варіантів практичної реалізації, які проявляються у вигляді шаблонів (моделей) проєктування. Найбільш поширеними варіантами на сьогодні являються трирівневі моделі групи MV [20]: MVC (Model View Controller), MVP (Model View Presenter) та MVVM (Model View ViewModel). Для реалізації автоматизованої системи створення супровідної документації освітнього процесу у вигляді веб-додатку буде використано MVC модель, оскільки її логіка є найближчою до розробленого проєкту архітектури.

Висновки

У процесі проєктування архітектури програмних чи програмно-апаратних систем можуть виникати різноманітні проблеми, які впливатимуть на кінцеву якість, ефективність, стійкість та надійність систем.

Недостатнє розуміння вимог до системи, неправильна декомпозиція, несумісність інтерфейсів, відсутність документації, проблеми з безпекою – це далеко не повний список основних проблем, які можуть виникати окремо або в поєднанні залежно від специфіки проєкту та умов розроблення. Вирішення цих проблем вимагає уважної аналітики вимог, глибокого розуміння прийнятих проєктних рішень та ефективного керування. Проте на сьогодні не існує чіткого та об'єктивного переліку дій, виконання яких призведе до створення якісної архітектури.

Саме тому в даній роботі було розроблено проєкт архітектури програмної автоматизованої системи створення супровідної документації освітнього процесу, яка включає в себе дві основні підсистеми: підсистема формування силабусів та підсистема формування індивідуальних планів.

Також проведено документування основних проєктних рішень для спрощення процесу розроблення та подальшого формування методології застосування архітектурних шаблонів (наприклад, MVC) для реалізації якісної трирівневої архітектури. У подальшому на базі створеного проєкту архітектури буде розроблено відповідну автоматизовану програмну систему.

Література

[1] N. Condori-Fernandez, P. Lago, "Characterizing the contribution of quality requirements to software sustainability," *Journal of Systems and Software*, vol. 137, pp. 289-305, 2018. DOI:10.1016/j.jss.2017.12.005.

[2] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2021.

[3] S. Ahmadisakha, V. Andrikopoulos, "Architecting for sustainability of and in the cloud: A systematic literature review," *Information and Software Technology*, vol. 171, 2024, 107459. DOI:10.1016/j.infsof.2024.107459.

[4] A. Jansen, J. Bosch, "Software architecture as a set

of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005, 6 November - 10 November, 2005*, pp. 109-120, 2005.

[5] Y. Cai, R. Kazman, "Software design analysis and technical debt management based on design rule theory," *Information and Software Technology*, vol. 164, 2023, 107322 DOI:10.1016/j.infsof.2023.107322.

[6] B. Bafandeh Mayvan, A. Rasoolzadegan, Z. Ghavidel Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *Journal of Systems and Software*, vol. 125, pp. 93-118, 2017. DOI:10.1016/j.jss.2016.11.030.

[7] M. O. Elish, M. A. Mohammed, "Quantitative analysis of fault density in design patterns: An empirical study," *Information and Software Technology*, vol. 66, pp. 58-72, 2015. DOI: 10.1016/j.infsof.2015.05.006.

[8] F. Al-Hawari, "Software design patterns for data management features in web-based information systems," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, is. 10, Part B, pp. 10028-10043, 2022. DOI:10.1016/j.jksuci.2022.10.003.

[9] C. Y. Baldwin, K. B. Clark, *Design Rules*, vol. 1: The Power of Modularity. Mit Pr, 2000.

[10] M. Tsai, S. Lee, "SW Shieh Strategy for Implementing of Zero Trust Architecture", *IEEE Transactions on Reliability*, vol. 73, is. 1, pp. 93-100, 2024. DOI: 10.1109/TR.2023.3345665.

[11] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, M. Picone, "Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories", *Computers in Industry*, vol. 149, 2023, 103918. DOI: 10.1016/j.compind.2023.103918.

[12] J. Fritzsch, J. Bogner, S. Wagner, A. Zimmermann, "Microservices Migration in Industry: Intentions, Strategies, and Challenges", in *Proc. 2019 IEEE International Conference on Software Maintenance and Evolution,* ICSME 2019, 30 September - 4 October, 2019, pp. 481-490.

[13] S. Khoshnevis, O. Ardestani, "Search-based approaches to optimizing software product line architectures: A systematic literature review", *Information and Software Technology*, vol. 170, 2024, 107446. DOI: 10.1016/j.infsof.2024.107446.

[14] V. Grassi, R. Mirandola, D. Perez-Palacin, "A conceptual and architectural characterization of antifragile systems", *Journal of Systems and Software*, vol. 213, 2024, 112051. doi:10.1016/j.jss.2024.112051.

[15] L. Xiao, Y. Cai, R. Kazman, "Design rule spaces: a new form of architecture insight", in *Proc. of the 36th International Conference on Software Engineering,* ICSE 2014, 1 May - 7 June, 2014, pp. 967-977. doi: 10.1145/2568225.2568241.

[16] С. Цибульник, Д. Бідник, & Д. Півторак, "Розроблення автоматизованої бібліографічної системи," *Вісник Національного технічного університету «ХПІ». Серія: Нові рішення у сучасних технологіях*, 2(12), с. 54–60, 2022. DOI: 10.20998/2413-4295.2022.02.08.

[17] K. Mrityunjay, C. Venkatesh, "Enhancing MVC architecture pattern description using its System of Systems model", in *Proc. of the 17th Innovations in Software Engineering Conference*, ISEC '24, Association for Computing Machinery, New York, NY, USA, Art. 8, pp. 1–11. DOI:10.1145/3641399.3641410.

[18] F. Tusa, S. Clayman, A. Buzachis, M. Fazio, "Microservices and serverless functions - lifecycle, performance, and resource utilisation of edge based real-time IoT analytics", *Future Generation Computer Systems*, vol. 155, pp. 204-218, 2024. DOI: 10.1016/j.future.2024.02.006.

[19] L. Carvalho et al., "On the Usefulness of Automatically Generated Microservice Architectures", *IEEE Transactions on Software Engineering*, vol. 50, no. 3, pp. 651-667, 2024. DOI: 10.1109/TSE.2024.3361209.

[20] С. Цибульник, Д. Зубарський, & Д. Півторак, "Прикладне програмне забезпечення для зберігання персональної інформації," *Вісник Національного технічного університету «ХПІ». Серія: Нові рішення у сучасних технологіях*, 1(15), с. 53–59, 2023. DOI: 10.20998/2413-4295.2023.01.07.

УДК 616.6:004.67

S. Tsybulnyk, V. Nakoryk, D. Pivtorak
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

DESIGN OF THE ARCHITECTURE OF THE AUTOMATED SYSTEM FOR CREATING THE ACCOMPANYING DOCUMENTATION OF THE EDUCATIONAL PROCESS

Background. The architecture of the automated system is the main source of quality of software and software-hardware systems. It determines how quickly and efficiently a developer is able to analyze, understand, test, extend, and maintain an automated system. Changes in the system architecture have a high cost due to its complexity and the possibility of destruction during expansion. At present, most of the knowledge and information about the design decisions on which the architecture is based are implicitly embedded in it, which leads to problems during the development process, in particular, during the programmatic implementation of its structural elements. It is determined that, despite the long way that the evolution of systems architecture has passed, today there are very few objective, repeatable and empirically

based methodologies and tools for the design and analysis of architecture. As a rule, software architects are programmers who have decades of experience in software development practice. Given their experience, they clearly understand that errors in the architecture design are the root causes of lower-level problems that manifest in the software code. Most often, a programmer feels when the architecture of his project is of low quality, because there are a large number of technical flaws, the number of which only increases with time. But most projects continue to be implemented, which leads to a decrease in the quality of the finished automated system.

Objective. The purpose of this work is to design and document the software architecture of the automated system for creating the accompanying documentation of the educational process in order to take the first step towards understanding the relationships and the impact of the adopted project decisions on the code base.

Conclusions. To achieve the goal, a class of multi-level architectures was chosen, among which the three-level architecture is the most common. Features of implementation of open and closed three-level architectures are considered. On the basis of the MVC architectural template, an architectural project of an automated system for creating accompanying documentation of the educational process has been developed. The main subsystems and data elements are documented.

Key words: automated system; architecture; software; MVC; documentation.

Надійшла до редакції
23 квітня 2024 року

Рецензовано
16 травня 2024 року