2/9(50). – С. 7–9. **3**. *Захарченко Н. В*. Оценка информационной скрытности таймерных сигнальных конструкций в системах передачи конфиденциальной информации/ *Н. В. Захарченко, В. В. Корчинский, Б. К. Радзимовский* // Збірник наукових праць ОНАЗ ім.О.С.Попова. – 2011. – № 1. – С. 3–8. **4.** *Захарченко Н. В.* Метод формирования сигнальных конструкций на основе хаотических и таймерных сигналов в системах передачи конфиденциальной информации/ *Н. В. Захарченко, В. В. Корчинский, Б. К. Радзимовский* // Збірник наукових праць ОНАЗ ім.О.С.Попова. –2011. – № 2. – С. 3–7. **5.** *Захарченко Н. В*. Многопользовательский доступ в системах передачи с хаотическими сигналами / *Н. В. Захарченко, В. В. Корчинский, Б. К. Радзимовский* // Восточно-Европейский журнал передовых технологий. – 2011. – № 5/9(53). – С. 26–29. **6.** *Корчинський В. В*. Повышение скрытности передачи на основе псевдослучайной перестройки рабочей частоты и таймерных сигналов / *В.В. Корчинський* // Вестник НТУ «ХПИ» – Харьков, 2012., – вип 66. – С. 63–67.

Надійшла до редколегії 20.03.2013

УДК 621.391

Конфиденциальная система связи на основе псевдослучайной перестройки рабочей частоты и таймерных сигналов/ Корчинский В. В. // Вісник НТУ «ХПІ». Серія: Нові рішення в сучасних технологіях. – Х: НТУ «ХПІ», – 2013. - № 1 (977). – С. 82-85. – Бібліогр.: 6 назв.

Для конфіденційної системи зв'язку багатокористувацького доступу запропоновано метод формування групового сигналу на основі псевдовипадкової перебудови робочої частоти і таймерних сигналів.

The method of forming group signal based on pseudo-random adjustment of operating frequency and timer signal is proposed for the multiuser access confidential communication system.

Keywords: timed signal pseudorandom restructuring frequency.

УДК 004.94

С. І. ШАПОВАЛОВА, канд. техн. наук, доц., НТУ «КПІ», Київ;
О. І. ЛАЗУРЕНКО, магістрант, НТУ «КПІ», Київ

ОПТИМІЗАЦІЯ АРХІТЕКТУРИ WEB-СИСТЕМ

Проведено дослідження існуючих на даний момент архітектур Web-систем, виявлено їх переваги та недоліки. Запропоновано удосконалення Onion-архітектури, що дає змогу зменшити використання ресурсів та підвищити її швидкодію.

Ключові слова: багаторівнева архітектура, Onion-архітектура, Web-архітектура, Web-системи, вирішення залежностей (dependency injection), інверсія контролю, рефлексія.

Введення. Проектування складних систем у вигляді Web-додатків з кожним роком займає все більшу частину на ринку програмного забезпечення. В наш час все більше компаній надають свої послуги через Інтернет. Як наслідок, користуються попитом програмні системи, створенні у вигляді Web-сайтів. Прикладом таких систем можуть бути Web-ресурси з надання кредитів, надання послуг обслуговування пасажирів, банківських послуг, внутрішні сайти компаній і системи менеджменту.

При проектуванні складних Web-систем необхідно враховувати наступні фактори:

- орієнтованість на довготривалий строк експлуатації;
- забезпечення взаємодії з джерелами ресурсів різного роду (бази даних різних типів, файлові системи, платіжні системи, використання сторонніх сервісів);
- стійкість до модифікацій після випуску завершеного продукту;
- легкість в нарощуванні функціоналу.

Розробка таких Web-систем потребує детального проектування на рівні створення концепції, правильного вибору архітектури та платформи для її реалізації. Якщо не поставитись до проектування архітектури Web-системи належним чином, то під час розробки можуть виникнути проблеми з тестуванням, а після випуску завершеної системи - труднощі в модифікації чи заміні модулів. Це стає причиною збільшення витрат часових

та людських ресурсів. Тому оптимізація архітектури складних Web-систем для зниження об'єму ресурсів, потрібних для розробки та експлуатації програмної системи, є **актуальним завданням**.

Мета роботи. Метою роботи є оптимізація Onion-архітектури для прискорення завантаження сторінок і зменшення ресурсоємності розробки складних Web-систем.

Аналіз останніх досліджень. Найбільш розповсюдженими архітектурами, що застосовуються на практиці, є:

- багатошарова архітектура (реалізована з використанням шаблону MVC (Model - View - Controler / Модель - Вигляд - Контролер) чи MVP(Model - View - Presenter / Модель - Вигляд - Презентер);
- onion-архітектура (може містити шаблон MVC на рівні інтерфейсу користувача).

Багатошарова архітектура виникла з міркувань розподілу коду на окремі частини для полегшення проектування та реалізації складних систем. Проте, незважаючи на переваги, великим недоліком такої архітектури є те, що бізнес- логіка тісно переплетена з сервісами доступу до зовнішніх джерел даних, таких як база даних, різні Web-сервіси та ін. Для ліквідації цього недоліку в 2008 році Джеффрі Палермо (Jeffrey Palermo) була запропонована Onion-архітектура [1]. Вона має ряд переваг у порівнянні з багатошаровою архітектурою:

- винесення бази даних у зовнішні сервіси;
- низька зв'язність рівнів системи;
- розподіленість між сервісами;
- можливість автоматичного тестування.

Концепція існуючих архітектур Web-систем. Традиційна багатошарова архітектура дозволяє проектувати Web-системи у вигляді двонаправленого ланцюга зображеного на рис. 1, [2].

Основним елементом такої архітектури є база даних, яка через рівень бізнес-логіки, логіки представлення та допоміжні сервіси взаємодіє з інтерфейсом користувача .

Багатошарова архітектура є хорошим рішенням для побудови Web-сайтів. Її можна реалізувати на будь-якій платформі та мові програмування. Вона дозволяє розділяти код програми на логічні фрагменти – рівні, що в свою чергу спрощує проектування програмної системи. Проте, незважаючи на зазначені переваги багатошарової архітектури, її застосування у великих проектах має ряд недоліків:

- всі рівні в даній архітектурі тісно пов'язані та часто переплітаються між собою;
- внаслідок міцного зв'язку між рівнями ускладнюється тестування окремих рівнів під час розробки програмної системи;
- весь додаток будується навколо певної бази даних, і при необхідності її заміни модифікації підлягає велика частина самої системи;
- добудовування нових сервісів завжди зачіпає проміжні рівні такої архітектури.

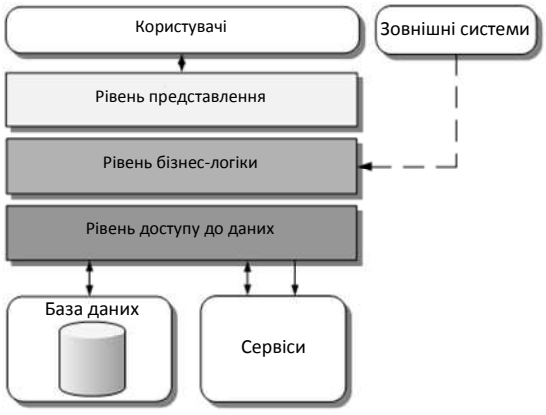

Рис. 1 - Схема багатошарової архітектури

Onion-архітектура дозволяє уникнути цих недоліків. Внаслідок використання інтерфейсів та технології вирішення залежностей DI (dependency injection) виходить мінімально зв'язний код, який легко піддається модифікації, тестуванню та дозволяє побудувати весь додаток навколо абстрактних сутностей моделі домену, а не навколо певної бази даних. В такій архітектурі база даних розміщена як сторонній ресурс. Концептуальна схема Onion-архітектури зображена на рис. 2, [3].

Особливістю такої архітектури є те, що будь-який зовнішній шар може безпосередньо викликати будь-який внутрішній шар. В традиційній багаторівневій архітектурі поточний шар може викликати тільки шар безпосередньо з ним пов'язаний. Це один з ключових моментів, який робить Onion-архітектуру відмінною від традиційної багаторівневої архітектури. Інфраструктура Onion-архітектури винесена в зовнішній шар, де немає прив'язаної до неї бізнес-логіки. Код, який взаємодіє з базою даних, буде реалізовувати інтерфейси ядра. Таким чином, можна модифікувати код в будь-якому зовнішньому шарі, не змінюючи нічого в ядрі додатку.

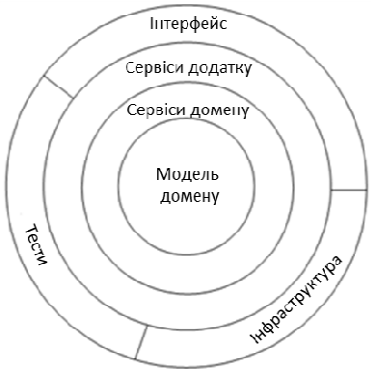

Рис. 2 - Схема Onion-архітектури

Тести в такій реалізації також винесені в зовнішній шар, тому не виникає труднощів у тестуванні будь-якого внутрішнього шару, включаючи інтерфейс користувача та код інфраструктури.

Такий підхід до архітектури додатків гарантує, що ядро додатку не змінитися при:

- зміні інтерфейсу користувача;
- зміні способу доступу до даних;
- модифікацій Web-служб;
- зміні методів введення-виведення інформації.

Onion-архітектура якнайкраще підходить для проектування великих і складних проектів. Вона дозволяє спроектувати систему таким чином, що модифікація коду впродовж життєвого циклу продукту вимагає мінімум затрат та зусиль. Реалізація цієї архітектури можлива лише на платформах, де підтримується технологія вирішення залежностей DI (Net, Java, PhP, Perl, Python).

Однак суттєвими недоліками Onion-архітектури є:

- неможливість автоматичної реєстрації всіх залежностей в DI контейнері і, як наслідок, недотримання чіткої залежності від верхніх шарів до нижніх;
- використання повільних DI контейнерів.

Реєстрація залежностей в Onion-архітектурі. Реєстрація залежностей в Onion-архітектурі, запропонованій розробниками [4], відбувається за схемою, зображеною на рис. 3. Для додавання модуля в Web-систему потрібно внести інформацію про новий інтерфейс та його реалізацію в модуль вирішення залежностей. Це створює додаткові залежності між рівнями архітектури та модулями. Ілюстрація цих залежностей показана на рис. 4.

Рис. 3 -Схема реєстрації залежностей при створенні нового модуля за традиційною Onion-архітектурою

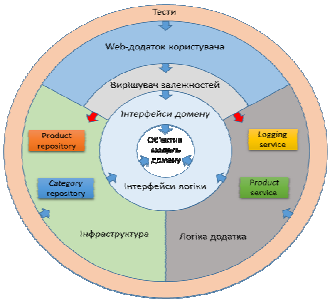

Рис. 4 - Схема залежностей між модулями в традиційній Onion-архітектурі

При такій реалізації Onion-архітектури залежність між проектами не виконується чітко від зовнішніх шарів до внутрішніх, що є своєрідним порушенням ідеології самої архітектури. До того ж, при великій кількості зовнішніх модулів у системі, реєстрація залежності способом безпосереднього внесення модифікацій до модуля вирішення залежностей створює додаткові незручності.

Для усунення цього недоліку пропонується скористатись рефлексією [5]. Вона

дозволяє дізнатись назви класів та інтерфейсів, не встановлюючи залежність на інші проекти. Переглянувши збірки проекту в автоматичному режимі, можна без проблем знайти визначений клас, який реалізує деякий інтерфейс та встановити залежність між ними. Щоб здійснити модифікації в Onion-архітектурі, потрібно внести зміни в модуль вирішення залежностей. Внаслідок цього процес додавання нового модуля буде мати вигляд, зображений на рис. 5.

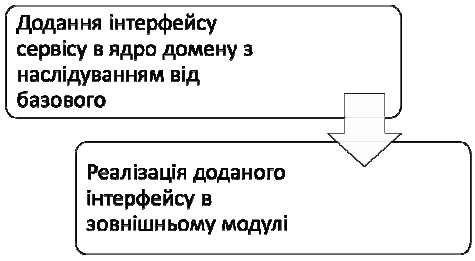

Рис. 5 - Схема реєстрації залежностей при створенні нового модуля за модифікованою Onion-архітектурою

За новою схемою при додаванні модуля не потрібно вносити зміни в модуль вирішення залежностей. Схема залежностей між проектами в модифікованій Onion-архітектурі зображена на рис. 6.

Лише додається інтерфейс в ядро програми та його реалізація до зовнішнього модуля. Залежності в модифікованій архітектурі чітко направлені від зовнішніх шарів до внутрішніх; відсутні залежності між модулями на одному рівні. Це дозволяє створювати модулі у вигляді окремих проектів дійсно незалежними один від одного. Вони можуть бути модифікованими чи заміненими на інші без втручання в код ядра, інтерфейсу користувача чи модуля вирішувача залежностей.

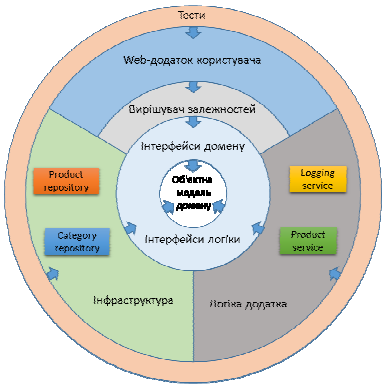

Рис. 6 - Схема залежностей між модулями в модифікованій Onion-архітектурі

В запропонованій розробниками реалізації Onion-архітектури [4], використано спосіб вирішення залежностей за допомогою Ninject [6], Autofac [7] та CastleWindsor [8] фреймворків, проте, як показують дослідження [9], вони є повільними, що негативно відображається на завантаженні сторінок. В цих же дослідженнях показано, що одним з найшвидших DI контейнерів є SimpleInjector. Залежності кожним з контейнерів були вирішені 1000000 разів. Результат вимірювання часу вирішення залежностей представлений в табл.1 [9] .

Таблиця 1 - Порівняння швидкості вирішення залежностей DI контейнерами

Контейнер	Час вирішення залежностей в різних режимах, мс		
	Одиночний (Singleton)	Тимчасовий (Transient)	Змішаний (Combined)
Без контейнера	74	70	80
AutoFac 3.0.1	827	2078	5192
Ninject 3.0.1.10	7745	21436	57453
SimpleInjector 2.0.2	75	102	106
Windsor 3.2.0	483	2465	7715

Щоб збільшити швидкодію вирішення залежностей в модифікованій Onion-архітектурі замість Ninject використаємо SimpleInjector. Таким чином, ліквідується другий недолік - використання повільних DI контейнерів.

Апробація модифікованої Onion-архітектури. Для доведення переваги запропонованих модифікацій Onion-архітектури було виконано такі завдання:

- створення сервісу отримання, редагування та збереження даних для порівняння ресурсоємності додавання модулів у систему (використано стандартну базу даних Northwind [10]);

- розробка тестової сторінки циклічних вирішень залежностей, для можливості наглядної демонстрації приросту швидкості завантаження Web-сторінок.

Тестова задача була випробувана на Onion-архітектурі розробників та модифікованій Onion-архітектурі. Вимірювався час розв'язання залежностей у відповідності до їх кількості. Результати випробувань наведені в табл. 2.

Таблиця 2 - Порівняння швидкості вирішення залежностей у звичайній та модифікованій Onion-архітектурі

Час залежностей вирішення, мс	Кількість			
	10	1000	10000	100000
Onion-архітектура, запропонована розробниками	6,0233	251,0324	2601,3427	25775,2364
Модифікована Onion-архітектура	1,0002	25,0208	328,0405	2380,2977

Використання SimpleInjector DI контейнера дозволило прискорити вирішення залежностей та швидкість завантаження сторінок, а автоматизація реєстрації залежностей – полегшити внесення модифікацій до системи.

Запропонована оптимізація Onion-архітектури була апробована при створені Web-системи моніторингу характеристик метеоумов на території розташування АЕС.

Висновки.

1. Проведено аналіз існуючих архітектур Web-систем. Обґрунтовано необхідність оптимізації Onion-архітектури.

2. Запропоновано модифікацію Onion-архітектури для зменшення залежностей між модулями, що дозволяє прискорити швидкодію роботи Web-системи, а також зменшує ресурси при розробці та модифікації системи .

3. Розроблено завдання обчислювальних тестів. Виконано тестові завдання для Onion-архітектури, запропонованої розробниками, та модифікованої Onion-архітектури.

4. Результатами проведених випробувань доведено ефективність використання модифікованої Onion-архітектури. Час розв'язання залежностей в модифікованій Onion-архітектурі на порядок менший, ніж в архітектурі, запропонованій розробниками.

Список літератури: 1. Jeffrey Palermo. The Onion Architecture [Електронний ресурс] : інформація / Дата оновлення: 29.07.2008. – Режим доступу: http://jeffreypalermo.com/blog/the-onion-architecture-part-1. **2.** *Alex Marvel*. Обзор и логическое разделение многослойных приложений. [Електронний ресурс] : інформація / Дата оновлення: 16.09.2012. – Режим доступу: http://brtrg.com/blog/post/154. **3.** Mirko Sertic. The Onion Architecture. [Електронний ресурс] : інформація / Дата оновлення: 18.03.2010. – Режим доступу: http://www.mirkosertic.de/doku.php/onionarchitecture. **4**. Tony Sneed. Peeling Back the Onion Architecture. [Електронний ресурс] : інформація /Дата оновлення: 8.10.2011. – Режим доступу: http://blog.tonysneed.com/2011/10/08/peeling-back-the-onion-architecture. **5**. *Джозеф Албахари*. C# 5.0. Справочник. Полное описание языка / *Джозеф Албахари, Бен Албахари*, 5-е издание, 1054 стр., ISBN 978-5-8459-1819-2, «ВИЛЬЯМС», 2013. **6**. Ninject - Open source dependency injector for .NET. [Електронний ресурс] : інформація – Режим доступу: http://www.ninject.org/. **7.** Autofac - An addictive .NET IoC container - Google Project Hosting. [Електронний ресурс] : інформація – Режим доступу: https://code.google.com/p/autofac/. **8**. Castle Windsor - Castle Project. [Електронний ресурс] : інформація – Режим доступу: http://docs.castleproject.org/Windsor.MainPage.ashx **9.** Daniel Palme. IoC Container Benchmark - Performance comparison [Електронний ресурс] : інформація / Дата оновлення: 13.09.2011. – Режим доступу: http://www.palmmedia.de/blog/2011/8/30/ioc-container-benchmark-performance-comparison. **10.** Northwind database [Електронний ресурс] : інформація – Режим доступу: http://northwinddatabase.codeplex.com/

Надійшла до редколегії 20.03.2013

УДК 004.94

Оптимізація архітектури WEB-систем/ Шаповалова С. І., Лазуренко О. І. // Вісник НТУ «ХПІ». Серія: Нові рішення в сучасних технологіях. – Х: НТУ «ХПІ», – 2013. - № 1 (977). – С. 85-90. – Бібліогр.: 10 назв.

Проведено исследование существующих на данный момент архитектур Web-систем, выявлены их преимущества и недостатки. Предложено усовершенствование Onion-архитектуры, что позволяет уменьшить использование ресурсов и повысить ее быстродействие.

Ключевые слова: многоуровневая архитектура, Onion-архитектура, Web-архитектура, Web-системы, разрешения зависимостей (dependency injection), инверсия контроля, рефлексия.

Investigated presently existing Web-system's architectures, detected their advantages and disadvantages. Proposed improvements of Onion-architecture, which can reduce the usage of resources and improve its performance.

Keywords: multilayer architecture, Onion-architecture, Web-architecture, Web-systems, dependency injection, inversion of control, reflection.

УДК 378.214.46

В. О. ЛІЩИНА, канд. техн. наук, доц., Луцький інститут розвитку людини Університету «Україна»

ПРОБЛЕМИ ФОРМУВАННЯ ЕТИЧНОГО КОДЕКСУ ІНЖЕНЕРІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В статті порушена проблема необхідності формування кодексу комп'ютерної етики, його складових елементів. Досліджено процес формування кодексів професійної діяльності фахівців програмного забезпечення, зокрема закордонний досвід. Проаналізовано основні засади формування етичних норми діяльності у сфері інформаційних технологій.

Ключові слова:фахівець ІТ-відділу, програмне забезпечення, кодекс етики, цінності, корпоративна культура, етичні норми, професійна діяльність.

Аналіз останніх досліджень і публікацій. Ще в 1972 році визначалося, що «проблема психологического изучения систем «человек-компьютер» носит не только междисциплинарный, но и межотраслевой характер» [4]. Це вивчення розвивалося по різних напрямах. Зокрема, вивчалася взаємодія людини і комп'ютера як «людина-знакова система» і створюваний при цьому образ світу [2, 6]. В цих дослідженнях підкреслювалося сприйняття комп'ютера як живої істоти,наділеної антропоморфними рисами. Так, розглядалися основні аспекти взаємодії людини і комп'ютера – анонімність,

регресія, створення стереотипів, реакції перенесення. Виділялися різні типи персоніфікації комп'ютера, пов'язані з глибинними людськими комплексами. З появою і розвитком мережі Інтернет багато досліджень присвячено феномену інтернет-залежності, взаємодії людей в мережі Інтернет. Розроблення психологічної тактики і стратегії створення програмних продуктів спиралися на дослідження, що розкривають структуру мислення людини при рішенні творчих задач.

Постановка завдання. У коло питань дослідження психології програмування входять діяльність програміста,психологічні закономірності прийому і перероблення інформації, функціонування психічних прийомів пам'яті і мислення у програміста, його працездатності, питання професійного відбору. Розвиток інформаційних технологій, зв'язаних із застосуванням персональних комп'ютерів, відбувається з великою швидкістю.

Але в цьому розвитку можна виділити окремі етапи, відповідні своєрідним поворотним моментам розв'язання кризи програмування і переходу на новий рівень.

Радикальним рішенням проблем кризи програмування по черзі оголошувалися пошук кращої мови програмування (1960-іроки), технології програмування (1970-і роки), інструментарію програмування (1980-і роки), систем якості (1990-і роки) [3].