https://doi.org/10.31891/2219-9365-2023-76-11
УДК 004.4

ФОРКУН Юрій
Хмельницький національний університет
https://orcid.org/0000-0002-7906-4191
forkun@ridne.net
МАРТИНЮК Валерій
Хмельницький національний університет
https://orcid.org/0000-0001-5758-4244
martynyuk.valeriy@gmail.com
ЯШИНА Оксана
Хмельницький національний університет
https://orcid.org/0000-0001-7816-1662
ksusha.ja@gmail.com

МЕТОД РОЗРОБКИ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРНОЇ СКЛАДОВОЇ ПРОГРАМНОГО ЗАСТОСУНКУ

У роботі досліджено питання вирішення проблеми ерозії архітектури та підтримки архітектури програмного забезпечення живою та актуальною. У цьому дослідженні було запропонуємо метод мінімізації або уникнення ерозії архітектури. Цей метод забезпечує систематичний спосіб проектування архітектури програмної системи. Він базується на концепції бачення архітектури, яка є ідеальним відображенням архітектури наявного програмного забезпечення. Цей метод розглядає архітектурний документ і проектний документ архітектури як два абсолютно різні документи. Крім того, це дослідження представляє спробу встановити взаємозв'язок між реалізованою архітектурою системи, проектним документом архітектури та баченням архітектури. Динамічна перевірка розробленого методу показує, що цей метод підходить для середніх і великих проектів, які мають кілька випусків. Усі дії цього методу обертаються навколо зусиль, щоб зберегти архітектуру програмного застосунку у відповідності з баченням архітектури. Ітераційний характер методу та синхронізація реалізованої архітектури з баченням архітектури допомагає виявити та зменшити ерозію архітектури. Цей метод не суперечить і не замінює будь-який процес або метод розробки чи управління (наприклад, RUP, метод гнучкої розробки, водоспадна модель тощо), але існує паралельно з ними.

Ключові слова: архітектура, алгоритм, програмний застосунок, технологія, структура.

FORKUN Yuriy, MARTYNYUK Valeriy, YASHINA Oksana
Khmelnytskyi National University

METHOD SOFTWARE ARCHITECTURAL COMPONENT DEVELOPMENT

This paper examines methods of addressing the problem of architecture erosion and keeping the software architecture live and up to date. In this study a method to minimize or avoid architecture erosion was proposed. This method provides a systematic way to design the architecture of the software system. It is based on the concept of architecture vision that is the ideal representation of the architecture of the software in hand. This method treats architecture document and architecture design document as two completely different documents. Moreover this study presents an effort to establish a relationship between the implemented architecture of the system, the architecture design document and an architecture vision. The dynamic validation of the devised method shows that this method is suitable for medium to large scale projects that have several releases. All the activities of this method revolve around the efforts to keep the architecture of the software aligned with an architecture vision. The iterative nature of the method and synchronization of the implemented architecture with the architecture vision helps to detect and reduce architecture erosion. This method does not conflict with or replaces any development or management process or method (like RUP, agile, water fall etc), but exists parallel to them.

Key words: architecture, algorithm, software application, technology, structure.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У теперішній час термін інженерія програмного забезпечення часто обмежується лише комп'ютерним застосунком. У справжньому розумінні це є не лише програмою, але й відповідною документацією та принципами проектування, необхідними для правильної роботи цих програмних застосунків. Програмні продукти можуть бути розроблені для конкретного клієнта або для загального ринку, тому вони зазнають серії думок та ідей, які пояснюють їх початкове створення, розробку, виробництво, експлуатацію, підтримку та зручність використання від одного покоління до іншого.

Таким чином, процес інженерія програмного забезпечення можна розглядати як набори дій і пов'язаних з ними результатів, які створюють програмний застосунок. Вони включають специфікацію програмного застосунку, розробку, валідацію та еволюцію. Модель процесу програмного застосунку представляє мережеву послідовність дій, об'єктів, перетворень і подій, яка втілює стратегії для досягнення еволюції програмного застосунку. Різні моделі процесу організовують ці дії по-різному, з різним рівнем деталізації, і вони найкраще підходять для проектів різної складності.

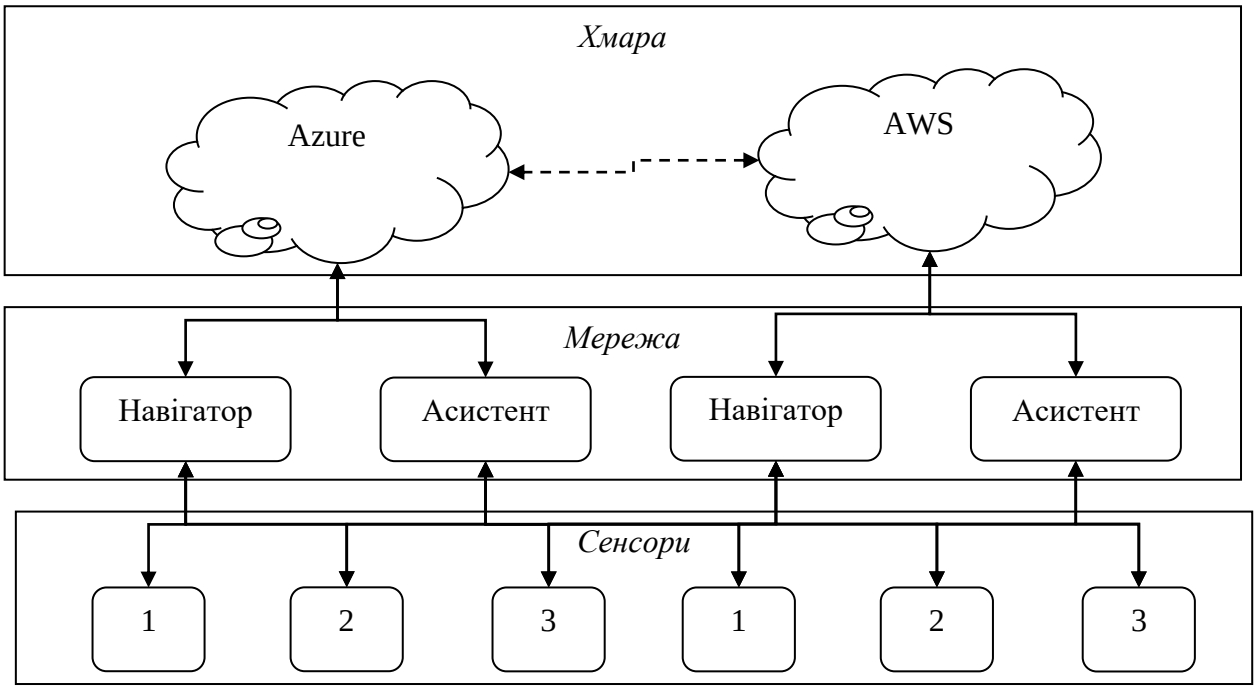

Рис. 1. Типова архітектура автономної системи без проміжного програмного застосунку та адаптивного менеджера реального часу

Практика проектування архітектури та методології програмного застосунку стала важливою частиною процесу проектування і є головною темою цієї статті. Архітектура програмного застосунку охоплює структури великих програмних систем. Архітектурний погляд на систему є абстрактним, у ньому відокремлюються деталі реалізації, алгоритму та представлення даних і зосереджується на поведінці та взаємодії елементів «чорного ящика». Архітектура програмного застосунку розробляється як перший крок до розробки системи, яка має набір бажаних властивостей, які можна представити за допомогою цієї формули (архітектура програмного застосунку = {Елементи, форми, обґрунтування/обмеження}).

Методологія програмного застосунку, з іншого боку, – це заздалегідь визначена послідовність подій, які необхідно виконувати та дотримуватися, щоб створити добре структурований і надійний програмний застосунок, який відповідає вимогам користувача та створює хороші тенденції масштабування.

Таким чином, у цій статті стверджується, що інженери-програмісти, які мають глибокі знання про архітектуру програмного застосунку та відповідну методологію для використання в процесі розробки програмного застосунку, будуть краще поінформовані, а отже, вироблятимуть якісний програмний застосунок та постачатимуть його у відповідний час, таким чином уникаючи розрив контракту, що є поширеним серед інженерів програмних застосунків.

О. М. Величко, О. В. Грабовський та Т. Б. Гордієнко [3] навели результати порівняльного аналізу для встановлення основних складових життєвого циклу програмного забезпечення для засобів вимірювальної техніки. Для цих досліджень використано як базові міжнародні стандарти щодо інженерії програмного забезпечення, так і специфічні міжнародні та регіональні документи щодо програмного забезпечення для засобів вимірювальної техніки.

О. В. Марковець та А. І. Синько [4] описали механізми формування якісної технічної документації до програмного забезпечення з урахуванням чинного законодавства.

У життєвому циклі розробки програмного застосунку зміни неминучі, це процес постійного моделювання та вдосконалення. Розробка архітектури програмної системи та написання вихідного коду не завершують життєвий цикл програмного застосунку. Програмна система розвивається зі змінами середовища, вимог і апаратного забезпечення. Ліенц і Свонсон [14] класифікували діяльність з підтримки та еволюції на чотири класи: адаптивна, досконала, коригувальна та превентивна. Адаптивна відноситься до змін у програмному середовищі, тоді як ідеальна покриває нові вимоги користувача; коригувальна – це виправлення помилок, а превентивна – запобігання проблемам у майбутньому [4].

По мірі того, як програмний застосунок розвивається, його розмір і складність зростають. В оптимальному випадку архітектура повинна масштабуватися для підтримки змін у вимогах. Якщо цими змінами не керувати належним чином, архітектура програмного застосунку погіршується. Архітектура програмного застосунку покликана керувати процесом його еволюції, одночасно розвиваючись.

Повідомлялося, що з розвитком програмного застосунку його архітектура занепадає. Ерозія архітектури визначається як явище, при якому початкова архітектура програмного застосунку довільно змінюється до точки, де вона більше не зберігає своїх ключових властивостей. Відповідно до Яктмана та ін. ерозія спричинена збільшенням складності архітектури, нечітким зв'язком між архітектурним документом та його реалізацією, збільшенням рівня дефектів та непередбачуваною поведінкою через модифікації. Є кілька проблем, які викликають ерозію. Більшість із цих проблем пов'язані зі способом розробки та обслуговування програмного застосунку. Ван Гурп і Бош визначають деякі з цих проблем, наприклад; більшість ітеративних методів розробки дозволяють включати нові вимоги, які впливають на архітектуру, складні нотації використовуються для розробки програмного застосунку, що ускладнює відстеження проектних рішень, і ці проектні рішення, як правило, суперечливі та збігаються. Крім того, розробники також приймають неоптимальні проектні рішення під час розробки та обслуговування. Окрім проблем у методах розробки, існує низка інших проблем, які спричиняють ерозію, таких як порушення опису архітектурного проекту під час впровадження, зміни в описі проекту без урахування його впливу, відхилення від початкової ідеї архітектурного проекту, невідповідність між описом проекту архітектури та його фактичною реалізацією .

Аналіз досліджень і публікацій

Наукові діячі сьогодення внесли значний вклад у розвиток методологій розробки архітектури програмного застосунку. Ряд досліджень був проведений для вирішення проблеми ерозії архітектури програмного застосунку. Ці дослідження пропонують такі підходи, як шаблони архітектури, мови представлення архітектури, методи розробки тощо.

За словами Перрі та Вольфа, оцінка та налаштування є двома факторами, які важливі для архітектури програмного застосунку. Супутня властивість еволюції - збільшення опору до змін, який зумовлений двома архітектурними проблемами: ерозією та дрейфом архітектури. Вони стверджують, що архітектурна ерозія є результатом порушень архітектури програмного застосунку. Архітектура програмного застосунку порушується через багато причин, як-от незрозуміла розробникам архітектура, часові обмеження, відсутність або мала кількість документації тощо. Ці порушення призводять до збільшення проблем у системі та сприяють опору змінам. Вони описують архітектурний дрейф як результат нечутливості до архітектури програмного застосунку. Ця нечутливість призводить до непристосованості архітектури до змін, у результаті чого архітектура втрачає узгодженість і ясність. Це збільшує ймовірність порушення архітектури, оскільки архітектурі бракує чіткості та вона стає складнішою. Вони запропонували модель програмної архітектури, яка складається з трьох компонентів, які є елементами, формою та обгрунтуванням. Ця модель акцентує увагу на архітектурних елементах даних, обробки та з'єднання. Крім того, вона також висвітлює їхні зв'язки та властивості.

Загалом ця ситуація носить назву старіння програмного застосунку. Стверджується, що старіння програмного застосунку неминуче, але є можливість уповільнити цей процес або через деякий час навіть змінити його наслідки. Крім того, загалом вважається, що причинами старіння програмного застосунку є:

– зміни в середовищі (домен, вимоги та технології) навколо програмного застосунку, і ці зміни не включені в програмний застосунок;

– зміни вносяться до системи недбало, не звертаючи уваги на їхній ефект (вплив змін), що погіршує систему;

– зміни не документуються належним чином, що призводить до ситуації, коли важко зрозуміти та внести зміни в систему.

У цьому дослідженні описується три основні недоліки старіння програмного застосунку:

– важкість внесення зміни або розширення програмного застосунку, що вже вийшов. Задля внесення змін, необхідно не тільки зрозуміти, які зміни потрібно зробити, але також де і як їх зробити;

– знижена продуктивність: у міру розвитку програмного застосунку його архітектура погіршується, а вимоги до якості, такі як час/простір, продуктивність знижується;

– зниження надійності: в результаті змін у програмному застосунку виникають помилки.

Щоб вирішити проблеми старіння програмного застосунку, пропонуються методи подолання або обмеження його наслідків:

– дизайн для успіху: програмні застосунки слід розробляти з урахуванням фактора змін. Досить важко передбачити фактичні зміни, але можна передбачити класи змін, наприклад заміну терміналу на новий тип або зміни в інтерфейсі користувача. Потім можливо організувати програмний застосунок так, щоб елементи, які, швидше за все, змінюватимуться, розміщувалися окремо. Принцип, який слід застосовувати, відомий під різними назвами, як-от «приховування інформації», «абстракція», «приховування даних», «розподіл інтересів», «орієнтація об'єкта» тощо;

– документація – ведення записів: разом із документацією коду принципи проектування і проектне рішення має бути задокументовано або записане у формі, яка може бути корисною в майбутньому.

Отримавши необхідну документацію, необхідно підтримувати її в актуальному стані, щоб вона відповідала системі;

– огляди – кілька думок: кожен дизайн та інші артефакти програмного застосунку повинні бути перевірені та затверджені особою, яка відповідає за довгострокове майбутнє продукту.

Зі збільшенням розміру та складності програмного застосунку починають проявлятися слабкі сторони існуючих програмних методів. Ван Гурп і Бош визначили, що ерозія дизайну спричинена низкою проблем, пов'язаних зі способом розробки програмного застосунку.

– простежуваність проектних рішень: дизайнерські рішення важко відстежити, оскільки використані позначки не мають виразності;

– збільшення вартості обслуговування: під час еволюції та процесу обслуговування розробники приймають неоптимальні проектні рішення. Для таких рішень може бути багато причин: або тому, що розробники не розуміють архітектуру, або тому, що більш оптимальне рішення вимагало б надто зусиль;

– накопичення проектних рішень: часто дизайнерські рішення накопичуються та взаємодіють таким чином, що якщо переглядається одне проектне рішення, також необхідно переглядати інші дизайнерські рішення;

– ітераційні методи: програмний застосунок розроблено з урахуванням очікуваних змін, але ітераційні методи дають змогу включити нові вимоги, які впливають на архітектуру;

– запити на зміни: це суперечить ітераційній природі багатьох методів розробки (екстремальне програмування, швидке створення прототипів тощо), оскільки ці методології зазвичай включають нові вимоги, які можуть мати вплив на архітектуру під час розробки, тоді як правильний дизайн вимагає знання про ці вимоги в заздалегідь.

У роботі [12] пропонують дві стереотипні стратегії для включення змін у програмний застосунок в ітераційні методи розробки.

– стратегія мінімальних зусиль. Ця стратегія заохочує максимально зберегти стару систему та коригує зміни в наступних ітераціях розробки. Перевага цієї стратегії полягає в тому, що кожна наступна ітерація призводить до відносно менших витрат;

– оптимальна стратегія проектування. Оновлення артефактів програмного застосунку шляхом внесення змін, необхідних для розробки оптимальної системи для нового набору вимог. Перевага цього підходу полягає в тому, що змінена система є оптимальною для вимог, оскільки будь-які суперечливі конструктивні рішення в попередній версії вирішені. Це означає, що майбутні зміни можна впроваджувати за відносно низьку вартість.

Постановка завдання

Метою роботи є дослідження методів розробки архітектури програмного застосунку задля підтримання його життєдіяльності та працездатності.

Викладення основного матеріалу

Метод V-BAM забезпечує дисциплінований підхід до планування діяльності з розробки архітектури програмного застосунку. Його мета полягає в тому, щоб забезпечити виробництво та еволюцію високоякісного програмного застосунку, який відповідає потребам кінцевих користувачів у межах передбачуваного графіка та зменшує або уникає ерозії архітектури. V-BAM полегшує розробку архітектури програмного застосунку, надаючи концепцію бачення архітектури, синхронізуючи архітектуру з реалізацією, підтримуючи рішення щодо проектування архітектури та цілі кожного випуску окремо, а також заохочуючи виконання дій, пов'язаних з архітектурою програмного застосунку, протягом кожного випуску.

Основними цілями V-BAM є:

– реалізувати ідею бачення архітектури на практиці шляхом впровадження систематичного ітеративного процесу розробки архітектури;

– допомагати в розробці та підтримці високоякісної архітектури програмного застосунку протягом усього життя програмної системи;

– структурувати зусилля, необхідні для реалізації наступного випуску програмної системи;

– зменшити та подолати ерозію та дрейф архітектури разом із пов'язаними з ними проблемами, щоб продовжити термін служби архітектури програмного застосунку (і самого програмного застосунку).

Архітектура в основному відображає вимоги до програмного застосунку. Важливо знати, що потрібно розробити, перш ніж фактично проектувати архітектуру. Передумовами методу V-BAM є:

– функціональні та якісні вимоги системи;

– крім того, слід визначити бізнес-правила та обмеження.

Якщо ці передумови не виконуються, це може негативно вплинути на бачення архітектури та архітектуру наступного випуску програмної системи. Якщо бачення спочатку було б розроблено на основі часткових або неправильних вимог, тоді було б не дуже корисно керувати архітектурою програмного застосунку.

Метод V-BAM складається з наступних п’яти кроків, як показано на рисунку 2. Ці кроки є послідовними, і вони відбуваються в одній послідовності для кожного циклу.

У реальності та практичному житті ці кроки значною мірою перекривають один одного, і важко провести межу між ними:

– визначити вимоги до архітектури – що має робити програмний застосунок. З технічної точки зору архітектура програмного застосунку в основному відображає вимоги до програмного застосунку. Важливо знати, що потрібно розробити, перш ніж фактично проектувати архітектуру. Основна мета або мета цього кроку полягає в аналізі та ідентифікації вимог до архітектури, вимог, які можуть вплинути на архітектуру програмного застосунку (на будь-якому рівні, будь-якою мірою);

Рис. 2. Схема методу на основі бачення та кроки для його реалізації

– розробити/уточнити бачення архітектури – бачення архітектури – це документ, який містить загальні цілі та задачі програмної системи. Архітектурне бачення являє собою ідеальну архітектуру системи, яка ідеально задовольняє функціональні та якісні вимоги. Крім того, він містить вказівки та мотивацію для архітектури в наступних випусках. Не важливо, чи реально зараз досягти цих цілей, важливіше мати ідеальні цілі, до яких потрібно прагнути. Бачення архітектури в основному виступає як специфікація системи, а проектний документ архітектури є втіленням цього бачення. Цілі, завдання тощо визначаються в баченні архітектури та реалізуються в різних випусках системи;

– розробити проектний документ архітектури конкретного випуску – основна мета цього кроку полягає в розробці документу проектування архітектури конкретного випуску, де загальна увага

приділяється випуску попереду в межах заданих ресурсів і часу. Вхідними даними цього кроку є бачення архітектури, архітектура попереднього випуску (якщо він існує) і вимоги до архітектури, специфічні для випуску. Результатом цього кроку є документ проекту архітектури для конкретного випуску. Цей крок починається після формування бачення архітектури та визначення вимог до архітектури для конкретного випуску. Проектний документ визначає шлях, на якому архітектура програмного застосунку повинна будуватися на наступному кроці;

– розробити архітектуру конкретного випуску – головна мета цього кроку – побудувати архітектуру програмного застосунку для конкретного випуску. На цьому кроці архітектура, яка була реалізована в попередньому випуску, змінюється на архітектуру, передбачену документацією щодо архітектури випуску. Архітектура програмного застосунку для конкретного випуску представлена в конкретній формі, яка керує розробкою програмного застосунку для конкретного випуску. Це реалізована форма архітектурного проектного документа, натхненна архітектурним баченням. Вхідними даними для цього кроку є пакет специфічних вимог до архітектури та документи щодо розробки архітектури програмного застосунку. У разі другого випуску й наступних вже мається документ про архітектуру та архітектура програмного застосунку попереднього випуску, як вхідні дані для цього кроку.

– синхронізувати архітектуру конкретного випуску з баченням і проектним документом архітектури – цей крок починається в кінці кожної ітерації, коли розробляється архітектура та навколо неї створюється програмний застосунок шляхом реалізації архітектури програмного застосунку. Щоб синхронізувати ці артефакти, необхідно визначити невідповідності на різних рівнях, як-от у реалізації, архітектурі програмного застосунку, проектному документі архітектури та баченні архітектури. Після виявлення невідповідностей треба оновити відповідні документи, щоб усунути ці невідповідності. Вхідними даними цього кроку є бачення архітектури програмного застосунку, проектний документ архітектури та архітектура програмного застосунку. Результатом цього кроку є синхронізована версія бачення архітектури програмного застосунку, документ проекту архітектури та документ архітектури програмного застосунку.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

У даній роботі було представлено метод розробки архітектури програмного застосунку. Він називається методом формування архітектури на основі бачення (V-BAM). Метод V-BAM є системним способом подолання/зменшення ерозії архітектури програмного застосунку. Він зосереджується на створенні ідеальної картини (бачення архітектури) архітектури програмного застосунку та зіставлення цієї ідеальної картини з реалізованою архітектурою, розробленою в попередньому випуску, задля того, щоб була архітектура програмного застосунку для наступного випуску. Основними вхідними даними методу V-BAM є специфікація вимог до програмного застосунку, бачення архітектури, проектний документ архітектури, архітектура конкретного випуску наступного випуску. Метод V-BAM складається з п'яти кроків. На першому етапі як функціональні, так і якісні вимоги системи ретельно аналізуються, щоб визначити вимоги рівня архітектури. На другому етапі архітектурне бачення розвивається/уточнюється. Архітектурне бачення являє собою ідеальну архітектуру системи, яка ідеально задовольняє функціональні та якісні вимоги. Це ідеальне зображення служить метою для натхнення. Не важливо, чи реально зараз досягти цих цілей, важливіше мати якісь ідеальні цілі, до яких потрібно прагнути. Коли у нас є вимоги до архітектури та бачення архітектури, на наступному кроці розробляється документ про дизайн конкретної архітектури, де основна увага приділяється поточному випуску в межах заданих ресурсів і часу. Цей проектний документ визначає шлях, за яким архітектура програмного застосунку має будуватися на наступному кроці. На наступному етапі розробляється конкретна архітектура випуску. Архітектура програмного застосунку для конкретного випуску представлена в конкретній формі, яка керує розробкою програмного застосунку для конкретного випуску. Це реалізована форма архітектурного проектного документа, натхненна архітектурним баченням. На останньому етапі методу синхронізується архітектуру конкретного випуску з баченням архітектури та проектним документом. Основна мета цього кроку полягає в синхронізації реалізації програмного застосунку з архітектурою програмного застосунку, проектним документом програмного застосунку та баченням архітектури. Синхронізація допомагає підтримувати ці артефакти у відповідності та актуальності, що допомагає подолати ерозію та дрейф архітектури.

Література

1. Maznan, Roslinda & Wan Kadir, Wan Mohd Nasir & Kadir, Wan.. A comparative evaluation of the three prominent approaches in adaptable software architecture, 2021.
2. Mall, R. Fundamentals of software engineering. PHI Learning Pvt. Ltd., 2018.
3. Величко, О. М. Особливості процесів життєвого циклу програмного забезпечення для засобів вимірювальної техніки / О. М. Величко, О. В. Грабовський, Т. Б. Гордієнко. Збірник наукових праць Одеської державної академії технічного регулювання та якості [Текст] / [редкол. : Коломієць Л. В. (голова) та ін.]. – Одеса : ОДАТРЯ, 2012 – Вип. 2 (17) 2020. – С .37-45

4. Марковець О. В. Формування якісної технічної документації до програмного забезпечення / О. В. Марковець, А. І. Синько // Вісник Вінницького політехнічного інституту. – 2021. – № 2. – С. 98-106. – Режим доступу: http://nbuv.gov.ua/UJRN/vvpi_2021_2_15.

5. Kumar, S., Rastogi, R. and Nag, R.,"Limitations of Function Point Analysis in Multimedia Software/Application Estimation", In Software Engineering, Springer, pp. 383-392, 2019.

6. Wani, Z.H. and Quadri, S.M.K., "Software Cost Estimation Based on the Hybrid Model of Input Selection Procedure and Artificial Neural Network", Artificial Intelligent Systems and Machine Learning,Volume 10, No 1, P.18-24, 2018.

7. Richards, M., Ford, N. Fundamentals of Software Architecture. O`Reilly, 2020.

8. Qusay I. Sarhan, Bestoun S. Ahmed, Miroslav Bures, Kamal Z. Zamli. Software module clustering: An in-depth literature analysis. IEEE Transactions on Software Engineering, 2020.

9. Philippe Kruchten, Robert Nord, Ipek Ozkaya. Managing technical debt: Reducing friction in software development. Addison-Wesley Professional, 2019.

10. Tushar Sharma. How deep is the mud: Fathoming architecture technical debt using designite. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), P. 59–60. IEEE, 2019.

11. Ruiyin Li, Peng Liang, Mohamed Soliman, and Paris Avgeriou. Understanding software architecture erosion: A systematic mapping study. Journal of Software: Evolution and Process, 2022.

12. Sandun Dasanayake, Sanja Aaramaa, Jouni Markkula, and Markku Oivo. Impact of requirements volatility on software architecture: How do software teams keep up with everchanging requirements? Journal of software: evolution and process, 2019.

References

1. Maznan, Roslinda & Wan Kadir, Wan Mohd Nasir & Kadir, Wan.. A comparative evaluation of the three prominent approaches in adaptable software architecture, 2021.

2. Mall, R. Fundamentals of software engineering. PHI Learning Pvt. Ltd., 2018.

3. Velychko, O. M. Osoblyvosti protsesiv zhyttievoho tsyklu prohramnoho zabezpechennia dlia zasobiv vymiriuvalnoi tekhniky / O. M. Velychko, O. V. Hrabovskyi, T. B. Hordiienko. Zbirnyk naukovykh prats Odeskoi derzhavnoi akademii tekhnichnoho rehuliuvannia ta yakosti [Tekst] / [redkol. : Kolomiiets L. V. (holova) ta in.]. – Odesa : ODATRIa, 2012 – Vyp. 2 (17) 2020. – S .37-45

4. Markovets O. V. Formuvannia yakisnoi tekhnichnoi dokumentatsii do prohramnoho zabezpechennia / O. V. Markovets, A. I. Synko // Visnyk Vinnytskoho politekhnichnoho instytutu. – 2021. – № 2. – S. 98-106. – Rezhym dostupu: http://nbuv.gov.ua/UJRN/vvpi_2021_2_15.

5. Kumar, S., Rastogi, R. and Nag, R.,"Limitations of Function Point Analysis in Multimedia Software/Application Estimation", In Software Engineering, Springer, pp. 383-392, 2019.

6. Wani, Z.H. and Quadri, S.M.K., "Software Cost Estimation Based on the Hybrid Model of Input Selection Procedure and Artificial Neural Network", Artificial Intelligent Systems and Machine Learning,Volume 10, No 1, P.18-24, 2018.

7. Richards, M., Ford, N. Fundamentals of Software Architecture. O`Reilly, 2020.

8. Qusay I. Sarhan, Bestoun S. Ahmed, Miroslav Bures, Kamal Z. Zamli. Software module clustering: An in-depth literature analysis. IEEE Transactions on Software Engineering, 2020.

9. Philippe Kruchten, Robert Nord, Ipek Ozkaya. Managing technical debt: Reducing friction in software development. Addison-Wesley Professional, 2019.

10. Tushar Sharma. How deep is the mud: Fathoming architecture technical debt using designite. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), P. 59–60. IEEE, 2019.

11. Ruiyin Li, Peng Liang, Mohamed Soliman, and Paris Avgeriou. Understanding software architecture erosion: A systematic mapping study. Journal of Software: Evolution and Process, 2022.

12. Sandun Dasanayake, Sanja Aaramaa, Jouni Markkula, and Markku Oivo. Impact of requirements volatility on software architecture: How do software teams keep up with everchanging requirements? Journal of software: evolution and process, 2019.