АКТУАЛЬНІ ПИТАННЯ ДИСЦИПЛІНИ «АРХІТЕКТУРА ПРОЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Манаков С. Ю.

кандидат технічних наук, доцент кафедри інформаційних технологій, Національного університету «Одеська юридична академія»

Архітектура програмного забезпечення має суперечливу репутацію в спільноті Agile [1]. Частково проблема полягає в тому, що архітектура є невдалою метафорою для результату роботи з розробки програмних систем. Натхнене роботою архітекторів-будівельників, слово викликає образи красивих дизайнів, які натякають на утопічне майбутнє. Але робота в галузі архітектури програмних систем набагато динамічніша, ніж це підтримує метафора будівельної архітектури. Будівлі статичні, і робота архітекторів будівель виконується лише один раз. Програмне забезпечення ж за своєю природою є постійно змінним і динамічним.

Аналогія з будівлею змусила деяких архітекторів програмного забезпечення занадто багато зосередитися на структурі та поведінці, а не на рішеннях, які створюють ці структури та поведінку. Справа не в тому, що структура та поведінка є неважливими, але вони є результатами процесу мислення, який важливо зберегти, якщо система хоче стабільно розвиватися з часом. Знати, чому хтось щось зробив, так само важливо, як знати, що він зробив. Те, що вони зробили, повинно бути помітним в коді, якщо він добре організований і прокоментований, але причина часто втрачається. Причина цього в тому, що архітектурні рішення в програмному забезпеченні рідко бувають чіткими. Майже кожне архітектурне рішення є компромісом між конкуруючими альтернативами, і важко побачити переваги альтернатив, доки ви не спробуєте декілька з них і не побачите, як вони працюють.

Це також одна з причин, чому архітектори програмного забезпечення все ще повинні бути розробниками; вони не можуть зрозуміти чи передбачити сили, що діють в системі, не розробивши й не перевіривши щось. Проектування вимагає достатнього знання системи, щоб сформулювати корисні гіпотези щодо атрибутів якості, а також досвіду написання коду та розробки тестів, які можуть оцінити ці гіпотези.

По правді кажучи, використання такої назви, як «архітектор програмного забезпечення», надає неправильне уявлення про характер роботи. Реальність така, що багато розробників програмного забезпечення виконують архітектурну роботу, вони просто не визнають її як таку. Щоразу, коли вони приймають рішення про те, як обробляти атрибути якості, вони впливають на архітектуру системи. Краще усвідомлення того, як неявні рішення впливають на здатність системи досягати цілей якості, є першим кроком у покращенні архітектури системи [2].

Архітектура сучасних програмних застосунків є в своїй основі дослідницькою діяльністю. Команди, які створюють сучасні застосунки, щодня стикаються з новими проблемами: безпрецедентними технічними проблемами [3], а також наданням клієнтам нових способів вирішення нових і відмінних проблем. Це безперервне дослідження означає, що архітектуру не можна визначити наперед, на основі минулого досвіду; команди повинні знайти нові шляхи задоволення вимог до якості.

Як приклад того, що дослідження є важливим для виявлення архітектури, розглянемо наступне: припустимо, що ви є частиною команди, яка працює над програмною системою, спочатку розробленою для обробки структурованих табличних даних, що зберігаються в базі даних SQL. Тепер цю систему потрібно вдосконалити, щоб обробляти неструктуровані дані, включаючи зображення та відео, і очікується, що обсяги значно збільшяться в порівнянні з тими, які зараз обробляє система. Ви розглядаєте можливість додати базу даних NoSQL до свого технологічного стеку для обробки нових типів даних, але оскільки ваша команда не має значного досвіду роботи з цією технологією, необхідно провести експерименти, щоб вибрати правильний продукт бази даних і налаштувати його відповідно до нових вимог до обсягу даних.

Коли команда працює над цими технічними проблемами, вони формують гіпотези про те, які підходи найкраще відповідатимуть їхнім бажаним QAR, які також є припущеннями та змінюватимуться з часом. Вони будують частину рішення для перевірки цих гіпотез і приймають рішення на основі результатів. Сукупним результатом цих рішень щодо відповідності QAR є архітектура системи. Команда може передавати ці рішення різними способами, в тому числі за допомогою документації та діаграм, але документи та діаграми – це не архітектура, а рішення та їхні причини.

Архітектура програмного забезпечення, як дисципліна, потребує оновлення. Її імідж страждає від багатьох старих уявлень про те, які проблеми їй потрібно вирішувати і як вона повинна вирішувати ці проблеми. Розгляд архітектури програмного забезпечення як безперервної діяльності, зосередженої на формуванні гіпотез про те, як система відповідатиме атрибутам якості, а потім використання емпіризму, щоб довести, що система їм відповідає, є сутністю безперервного підходу до архітектури програмного забезпечення. Також потрібно віддалити питання архітектури від людей, які не мають відношення до розробки, і передати її в руки людей, які насправді можуть зробити її реальною та виконаною, – розробників. Тільки тоді стає можливим досягнути високої стабільності, якої вимагають сучасні застосунки.

Список використаних джерел:

1. Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice. SEI Series in Software Engineering. 3rd edition. 2012. 640 p. ISBN 978-0321815736.

2. Роберт Мартін. Чиста архітектура: мистецтво розробки програмного забезпечення. Вид-во: Фабула. 2019. 416 с. ISBN 978-617-09-5286-8.
3. Марк Річардс. Основи архітектури програмного забезпечення: інженерний підхід. O'Reilly Media. 2020. ISBN 9781492043454.

ТЕНДЕНЦІЇ ВИКОРИСТАННЯ JAVASCRIPT FRAMEWORKS

Мироненко А. А.

студент 1-го курсу факультету кібербезпеки та інформаційних технологій Національного університету «Одеська юридична академія»

Для того щоб залишатись конкурентоспроможним на ринку IT-праці, кожен працівник IT-індустрії має бути обізнаним із сучасними трендами вебтехнологій. Особливо затребуваними нині на ринку IT-праці є веброзробники. Так, згідно зі статистикою Internet Live Stats нині у світі активними є 1.9 мільярдів вебсайтів [1]. Інтерактивна взаємодія сучасних вебсайтів з користувачами забезпечується засобами вебпрограмування, тому веброзробники мають бути на крок попереду у вивченні тенденцій розроблення функціонального користувальницького інтерфейсу сайтів.

Оскільки бекенд і фронтенд вебсайту реалізуються різними програмними засобами, то до бекенд-програмістів та до фронтенд-програмістів висуваються вимоги володіння відповідними засобами їх розробки. Так, фронтенд-програміст (Front-End Developer / Front-End Programmer), крім верстки, має володіти навичками розробки клієнтських скриптів, наприклад, мовою програмування JavaScript (JS) та знати принаймні один із JavaScript-фреймворків: Vue.js, Angular, React Native, Ember.js, Meteor або Backbone.js [2].

Що стосується мови JavaScript (або JS), наразі вона є однією з найперспективніших мультипарадигмальних мов програмування. За результатами проведеного опитування 2021 року на форумі з програмування Stack Overflow 64,96% з 83 052 опитаних професійних розробників віддали перевагу саме JavaScript [3].

Завдяки використанню наборів бібліотек JS-коду – JavaScript-фреймворків, завдання веброзробника полегшується в рази. Набори різноманітних інструментів цих фреймворків позбавляють програмістів від рутинних дій та суттєво економлять час, оскільки на основі шаблонів дозволяють формувати оптимальний код для розв'язання типових задач веброзробки.

Серед численних JavaScript-фреймворків за результатами опитування, проведеним State of Frontend 2021, в якому взяли участь понад 4,5 тисячі професійних розробників, в трійку найпопулярніших входять [4]:

React – 74,2%;

Angular – 33,4%;

Vue.js – 29,9%.

Для оцінки JS-фреймворків використовують різні підходи та метрики,