АДАПТИВНА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (ASD)

Васюта Василь Васильович
к.т.н., доцент
Барсуков Станіслав Георгійович
студент
Національний університет Полтавська політехніка
імені Юрія Кондратюка
м. Полтава, Україна

Вступ. /Introductions. Адаптивна розробка програмного забезпечення (ASD) – це методологія розробки, створена Джимом Хайсмітом та Семом Байєром, дуетом менеджерів проектів. ASD стверджує, що правильний спосіб досягти успіху, коли справа доходить до складних програмних проектів, – це постійно вчитися протягом усього проекту, не дотримуючись жорстких планів.

Мета роботи. /Alim. Розглянути сутність адаптивної розробки програмного забезпечення (ASD).

Матеріали та методи. /Materials and methods. Для досягнення поставленої мети у роботі використані методи теоретичного узагальнення та аналізу.

Результати та обговорення. /Results and discussion. В ASD проекти реалізуються в ітеративному циклі, що складається з трьох фаз (рис. 1), що перекриваються: спекуляції, співпраці та навчання. Фаза "спекуляції" (Speculate) – це те, що зазвичай називають "плануванням" в інших методологіях, гнучких чи ні. Він відкрито визнає парадокс складання планів у складному сценарії, що швидко змінюється.

Етап співпраці (Collaboration) показує важливість тісної та постійної співпраці не тільки всередині команди розробників, але також між розробниками та кінцевими користувачами програмного забезпечення [1].

Етап навчання (Learn) показує, що всі учасники процесу розробки програмного забезпечення постійно навчатимуться протягом усього проекту.

ASD, як підхід, орієнтований на високу швидкість використовує такі

методи, як обмежені за часом ітераційні цикли, планування з урахуванням ризиків та паралелізм для досягнення швидкої доставки цінності для клієнта, а також враховує невизначеність та майже хаос, які притаманні складним проектам з високим ризиком.

В ASD, як і більшості гнучких методологій, робота виконується циклами чи ітераціями. Під час таких циклів розробники як створюють нові компоненти, а й вносять необхідні зміни у вже існуючі [2].

Ключова різниця між адаптивною розробкою програмного забезпечення та іншими методологіями полягає в тому, що в ASD цикли базуються на компонентах, а не на завданнях. Наприклад, команди, які використовують інші методології, часто розбивають власні історії на більш деталізовані завдання, які потім доручаються розробникам для реалізації. В ASD у центрі уваги завжди знаходиться бажаний результат. Він визначає компоненти як групу функцій, що плануються, реалізуються і поставляються разом.

Компоненти в значенні ASD відносяться не тільки до візуальних компонентів, таких як кнопки, поля форми або інші елементи графічного інтерфейсу. Він також включає всі пов'язані "речі", які реалізовані разом.

У ASD є три типи компонентів: основні компоненти, технологічні компоненти та допоміжні компоненти. Первинні компоненти відносяться до самих бізнесфункцій. З іншого боку, технологічні компоненти складаються з частин технології або інфраструктури, які мають бути в наявності для реалізації основних компонентів. До технологічних компонентів відносяться не тільки обладнання та мережева інфраструктура, але також програмне забезпечення, таке як операційні системи, бази даних, фреймворки та багато іншого. Часто буває, що такі компоненти готові до використання. Якщо це не так, їх необхідно призначити циклам установки [2].

Компоненти підтримки включають решту, що не передбачено двома іншими типами. Наприклад, зовнішня документація, навчальні матеріали та багато іншого. Як і в інших методологіях, цикли в ASD обмежені за часом не лише на рівні кожного окремого циклу розробки, але й на рівні проекту.

Нарешті, цикли пов'язані з ризиком та толерантні до змін.

Ризико-орієнтованість можна розуміти як готовність швидко зазнати невдачі, поміщаючи елементи підвищеного ризику до більш ранніх циклів. Якщо ви визначили ризик у своєму проекті, вам слід спробувати зменшити ймовірність невдачі. Однак, якщо зниження ймовірності неможливе, ASD рекомендує прискорити виникнення відмови. Причина проста: якщо це рішення приречене на провал, краще і дешевше дізнатися про це раніше, ніж пізніше.

Толерантність до змін означає, що замість того, щоб припускати, що більшість дій відбуватиметься за планом, ви почнете з передумови, що все змінюватиметься, причому часто. В ASD команди створюють середовище, стійке до змін, щоб краще керувати змінами. Один із способів досягнення цього скорочення часових рамок. Крім того, ASD просуває культуру, в якій розробники постійно ставлять собі питання про зміни, а також спостерігають та оцінюють, що роблять конкуруючі послуги та продукти [21]. Замість того, щоб намагатися уникнути змін, ASD приймає це на благо клієнта.

Адаптивний життєвий цикл можна розглядати як еволюцію еволюційного життєвого циклу, так званого спіральним життєвим циклом, який почав виявлятися у середині 1980–х років. Недоліком еволюційного життєвого циклу є те, що багато його послідовників не відмовилися від свого старого мислення, заснованого на передбачуваності [1]. Адаптивний життєвий цикл усуває ці недоліки, відбиваючи внутрішню невизначеність розробки складних програмних систем. Це відображається, наприклад, у зміні назв фаз, але виходить за рамки цього. Адаптивний життєвий цикл також потребує зміни стилю керування (рис. 2).

Фаза «спекуляції» в ASD звертається до внутрішнього феномена, що виникає при плануванні складних сценаріїв. Оскільки ви не можете передбачити свої результати, на етапах планування ви з більшою ймовірністю помилитеся, ніж маєте рацію. ASD визнає це, позбавляючись слова "планування", яке є завантаженим терміном, що передбачає наявність великої кількості речей, і замінює його словом "спекулювати". Спекулюючи команди

виконують наступне:

- визначають свою місію в міру своїх можливостей;
- висловлюють загальне уявлення про свої кінцеві цілі, що досягається шляхом створення артефактів місії та обміну ними;
- приймають механізми, які допомагають їм адаптуватися та змінюватися в міру навчання протягом усього проекту.

Артефактів місій в ASD в основному три:

- хартія бачення проекту. Просте візуальне резюме проекту, яке включає бізнес–результати та основні ринкові відмінності;
- паспорт проекту. Короткий документ, що узагальнює важливу інформацію для проекту, яку команда та замовники можуть використовувати як орієнтир;
- короткий опис продукту. Високорівневий перелік специфікацій продукту, основна мета якого – визначити очікувані компоненти – групу функцій – для продукту.

Фаза спекуляції ділиться на два етапи: етап ініціювання проекту та етап адаптивного планування [3].

Етап ініціювання проекту – це ранній етап, на якому організація розробляє інформацію про управління проектом, формулювання місії проекту та початкових вимог.

Етап адаптивного планування – це етап, на якому компоненти програмного забезпечення призначаються для збирання або збирання.

Останній крок є необов'язковим, тому що в ідеалі адаптивне управління має відмовитися від контролю та мікроменеджменту, воліючи визначати «що» та не дбати про специфіку «як». Але оскільки перехід до адаптивного стилю може бути важким і навіть лякаючим, для згладжування такого переходу можна використовувати перелік завдань проекту.

Етап спільної роботи – це етап, на якому розробники фактично виконують роботу з розробки. Цей етап поділено на цикли, і кожен із них приносить користь клієнту.

Найважливішою і часто упущеною з уваги характеристикою команд, які добре справляються зі складними проектами, є високий технічний рівень кожного члена. Не має значення, якщо команди добре взаємодіють та добре спілкуються, якщо технічні навички, які кожен учасник привносить до команди, не підходять для цього завдання [4]. Крім того, команди мають бути невеликими, менш як 10 осіб. Це дуже важливо, тому що відмінна співпраця потребує інтенсивної взаємодії, яка ускладняється зі зростанням команди.

ASD стверджує, що команда повинна мати всі навички, необхідні для виконання роботи. Це також притаманно іншим гнучким методологіям, наприклад. концепція "всієї команди" від екстремального програмування. Однак ASD наголошує, що необхідне поєднання навичок має включати не лише технічні навички, але також навички міжособистісного спілкування та бізнесу.

Під час розробки адаптивного програмного забезпечення команди працюють відповідно до чотирьох основних цінностей: взаємна довіра, взаємна повага, взаємна участь та взаємні зобов'язання. Взаємна довіра має бути досягнута в команді через чесність – кажучи правду; безпека – створення комфортного, гостинного середовища для вираження ідей; та надійність знаючи, що ви можете покластися на своїх товаришів за командою.

Ввічливість – це мета великих команд, але не обов'язково гармонія. Конфлікт неминучий, коли є різноманітність, а за правильного вирішення він підвищує творчий потенціал. У командах, які використовують адаптивну розробку програмного забезпечення, всі повинні поважати та цінувати за внесок, який вони вносять у проект, незалежно від областей, у яких вони спеціалізуються. Відкриті спільні команди працюють найкраще, коли у них бере участь більше людей. Цінність взаємної участі полягає у визнанні того, що люди мають різний рівень досвіду та знань, а також знання у різних галузях.

Цінність полягає не в рівній участі, а у взаємній участі. Кожна людина в команді може робити свій внесок і виражати себе відповідно до своїх уподобань і навичок, але вся участь цінується однаково.

Цінності спираються одна на одну. Неможливо досягти високого ступеня взаємної участі, якщо команда ще не має взаємної довіри. Довіра потрібна для того, щоб члени команди відчували себе в безпеці та комфортно висловлюючи свої ідеї, не побоюючись засудження чи глузувань [5]. Члени команди однаково прихильні до мети проекту і несуть відповідальність за його успіх. Навіть якщо успішні програмні проекти мають один або навіть кілька лідерів, це не означає, що інші члени команди звільнені від відповідальності або прихильності до кінцевого результату. Навпаки: в адаптивній розробці програмного забезпечення результат належить кожному, і відмова від особистої відповідальності може і часто призводить до катастрофи.

Етап навчання – це по суті те, що інші методології називають оглядом або ретроспективою. На цьому етапі розроблений продукт аналізується з технічного погляду, а також з погляду клієнта. Крім того, робота команди також перевіряється у пошуках можливостей для покращення.

Висновки. /Conclusions. ADS, як методологія, дає чітке бачення парадоксу планування: оскільки коли справа доходить до складних проектів, ви не можете точно знати, куди ви потрапите.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Adaptive Software Development Explained [Електронний ресурс] Режим доступу до ресурсу: https://www.laneways.agency/adaptive-software-development-explained/.

2. Adaptive Software Development - Lifecycle [Електронний ресурс] Режим доступу до ресурсу: https://www.tutorialspoint.com/adaptive_software_development/adaptive_software_development_lifecycle.htm.

3. Філдінг Пол Джон. Як керувати проєктами. Основні навички проєктного менеджменту: вчасні результати в межах бюджету. Фабула. 2020. 240 с.

4. Bender J. Professional Test Driven Developmen, 2011. 536c.

5. Rajesh R. V. Becoming an Agile Software Architect / Rajesh., 2021. 372c.

6. Лагунова І. А. Сутність та принципи концепції ризик-менеджменту. Актуальні проблеми державного управління. 2018. № 1 (53). С. 44–52.