Лабораторна робота 2

Методології управління ІТ-проєктами

Мета роботи: знайомство з методологіями управління ІТ-проектами.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні положення

В даний час управління проектами має свою методологію і ґрунтується на певних стандартах. В основі таких методів лежать методики мережевого планування, які з'явилися наприкінці 1950-х рр. США. У цьому методики управління проектами широко поширилися у країнах із ринковою економікою, а й у країнах із т. зв. "планової" економікою. Вони почали використовуватися в будівництві, що послужило основою їхнього поширення в інших галузях та появі методів проектного управління.

Прикладом тут може бути об'єднання зусиль фірми «Дюпон» та фірми «Ремінгтон Ренд» для складання плану графіка комплексних робіт з модернізації заводів «Дюпон» за допомогою обчислювальної машини Univac у 1956 р. Результатом стало створення раціонального та простого методу опису проекту на ЕОМ – методу критичного шляху (CPM – Critical Path Method).

Практично паралельно та незалежно було створено метод аналізу та оцінки програм PERT (Program Evaluation and Review Technique) у військово морських сил США. Цей метод розроблявся корпорацією «Локхід» консалтинговою фірмою Буз, Аллен енд Гамільтон для ракетної системи Поларіс. Проект включав 3800 основних підрядників та складався з 60 000 операцій. Завдяки використанню даного методу, проект став успішним і завершився на два роки раніше терміну. Після наочного успіху метод PERT став використовуватись у збройних силах США для планування проектів.

Завдання: знайдіть приклади інших великих успішних проектів минулого століття, які використовують для управління.

З одного боку, застосування інформаційних технологій для планування та управління проектами давало на той час значну перевагу, а з іншого боку, перші комп'ютери були дорогими і залишалися доступними лише великим компаніям. Усе змінилося з появою персональних комп'ютерів, тепер комп’ютер став робочим інструментом керівників ширшого кола.

Система управління проектами постійно розвивалася і стала самостійною галуззю професійної діяльності. У результаті було створено уніфіковані методології, інструментарії, механізми, стандарти, доступні й у проектів у IT-сфері. Наприклад, на сьогоднішній день існує єдина Міжнародна асоціація управління проектами – IPMA (International Project Management Association) із центром у м. Цюріх (Швейцарія).

З найпоширеніших можна відзначити процесну модель, що використовується в документах методологічних засад управління проектами - Project Management Body of Knowledge (PMBOK) Американського інституту управління проектами (PMI). Цей документ визнається міжнародним стандартом де-факто. Крім того, стандарт ISO 10006:1997 надав ряду найважливіших положень PMBOK статусу стандарту де-юре.

В 2014 р. вийшло п'яте видання PMBOK, що містить вказівки на 589 сторінок. Усі положення представлені на сайті http://www.pmi.org/default.aspx.

Дані Міжнародної асоціації управління проектами (IPMA) свідчать, що використання сучасних методологій управління проектами заощаджує близько 20–30% часу та близько 15–20% коштів на здійснення проектів.

Усі методології (ще їх називають моделями, методиками) розробки програмного забезпечення класифікують за «вагою», тобто за кількістю формалізованих процесів та детальністю їхньої регламентації. Отже, що більше процесів документовано, що більш детально описана методолгія, то більше вписуватиметься її «вага».

1. **Важковагові методології**:

– Capability Maturity Model for Software (SW-CMM) визначає п'ять рівнів зрілості проекту;

– Rational Unified Process (RUP) – ітеративна модель розробки;

– Microsoft Solutions Framework (MSF) – база знань компанії Microsoft з розробки програм;

– Personal Software Process – модель визначає вимоги до компетенцій розробника. Team Software Process – модель орієнтує на самоврядні команди від 3 до 20 розробників.

2. **Легкі або Agile- методології**:

– eXtreme Programming або XP – екстремальне програмування, що пропонує 12 інженерних практик;

– Crystal Clear – сімейство методологій, що визначають необхідний ступінь формалізації процесу розробки залежно від кількості учасників та критичності завдань;

– Feature Driven Development (FDD) - функціонально-орієнтована технологія;

– OpenUP – ітеративно-інкрементальний метод розробки програм, що позиціонується як легкий та гнучкий варіант великовагової методології RUP;

– Scrum – управління розробкою інформаційних систем із високим ступенем невизначеності;

– Kanban - методологія "ощадливого виробництва".

Результати дослідження Agile Survey щодо популярності гнучких методологій представлені на рис. 1 [11].

Легкі методології Agile з'явилися порівняно недавно. У лютому 2001 р. 17 фахівців (консультантів та практиків) провели семінар, на якому сформулювали основні засади гнучкої розробки програмного забезпечення –

Agile Manifesto – маніфест гнучкої розробки. Він перекладений багатьма мовами світу і доступний на сайті http://agilemanifesto.org/iso/.

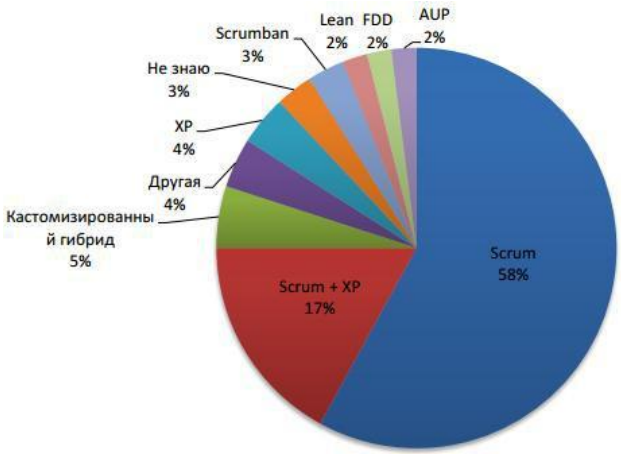

Рис. 1. Популярність гнучких методологій

Ідея всіх гнучких (легковагових) методологій полягає в тому, що процес, що застосовується в розробці, повинен бути адаптивним, орієнтованим на людей та їх взаємодію. По суті, це швидше набір практик, ніж методологія.

На цьому ж семінарі було запропоновано нову назву таких методологій – гнучка розробка Agile Software Development.

При виборі моделі управління проектом можна орієнтуватися на таку таблицю (Таблиця 1).

А. Коуберн, один із авторів «Маніфесту», провів аналіз IT-проектів за останні 20 років [4], виконаних на підставі різних моделей управління: від полегшених, гнучких до важких (CMM-5). Даний аналіз показав відсутність кореляції між успіхом чи провалом проектів та обраними моделями розробки, що застосовуються у проектах.

А. Коуберн зробив також висновок про те, що:

1. У кожного проекту має бути своя модель процесу розробки.
2. Кожна модель має свій час.

Таблиця 1 - Вибір методології управління проектом

Вага моделі	Переваги	Недоліки
Важковагові	Процеси розраховані на середню кваліфікацію виконавців; Велика спеціалізація виконавців; Низькі вимоги до стабільність команди; Відсутні обмеження по обсягу та складності виконуваних проектів.	Ввимагають суттєвої управлінської надбудови; Тривалі стадії аналізу та проектування; Формалізовані комунікації.
Легкі (Гнучкі)	Зниження непродуктивних витрат, пов'язаних з управлінням проектом, ризиками, змінами, конфігураціями; Спрощення стадій аналізу та проектування, основний акцент на розробку функціональності, поєднання ролей; Неформальні комунікації.	Ефективність сильно залежить від індивідуальних здібностей, вимагають більш кваліфікованої, універсальної і стабільної команди; Об`єм і складність виконуваних проектів обмежені.

Для розробників це означає, що немає єдиного правильного процесу розробки.

Уніфікований процес Rational (Rational Unified Process, RUP) [4] є досить складною, детально пропрацьованою ітеративною моделлю життєвого циклу ПЗ. RUP є програмним продуктом в якості доповнення до мови моделювання UML, розроблений компанією Rational Software, яка нині входить до складу IBM.

Основні принципи RUP

Історично RUP є розвитком моделі процесу розробки, прийнятої в компанії Ericsson в 70-х-80-х роках XX століття. Ця модель була створена Джекобсоном (Ivar Jacobson), який у 1987 заснував власну компанію Objectory AB саме для розвитку технологічного процесу розробки ПЗ як окремого продукту, який можна було б переносити в інші організації. Після вливання Objectory в Rational в 1995 розробки Джекобсона були інтегровані з роботами Ройса (Walker Royce, сина автора «класичної» каскадної моделі), Крухтена (Philippe Kruchten) і Буча (Grady Booch), а також з тим, що розвивалося паралельно універсальною мовою моделювання (Unified Modeling Language, UML) [17].

Основними принципами RUP є:

1. Ітераційний і інкрементний (нарощуваний) підхід до створення ПЗ.
2. Планування і управління проектом на основі функціональних вимог до системи – варіантів використання.
3. Побудова системи на базі архітектури ПЗ.

Перший принцип є визначальний. Відповідно до нього розробка системи виконується у вигляді декількох короткострокових міні-проектів фіксованої тривалості (від 2 до 6 тижнів), що називаються ітераціями. Кожна ітерація включає свої власні етапи аналізу вимог, проектування, реалізації, тестування, інтеграції і завершується створенням працюючої системи.

Фази проекту

З точки зору етапів (фаз) проектування, то вони, загалом, аналогічні «водоспаду». Але RUP виділяє в життєвому циклі 4 основних фази, у рамках кожної з яких можливе проведення декількох ітерацій.

1. Фаза початку проекту (Inception). Основна мета цієї фази – досягти компромісу між усіма зацікавленими особами відносно завдань проекту і ресурсів, що виділяються на нього. На цій фазі визначаються основні цілі проекту, керівник проекту і бюджет проекту, основні засоби його виконання – технології, інструменти, ключові виконавці, а також відбувається, можливо,

апробація вибраних технологій з метою підтвердження можливості досягти цілей з їх допомогою. На цю фазу може йти близько 10% часу і 5% трудомісткості одного циклу.

Результатами початкової стадії є:

- загальний опис системи: основні вимоги до проекту, його характеристики і обмеження;
- початковий бізнес-план;
- план проекту, що відбиває стадії і ітерації;
- один або декілька прототипів.

2. Фаза проектування (Elaboration). Основна мета цієї фази – на базі основних, найбільш суттєвих вимог розробити стабільну базову архітектуру продукту, яка дозволяє вирішувати поставлені перед системою завдання і надалі використовується як основа розробки системи. На цю фазу може йти близько 30% часу і 20% трудомісткості одного циклу.

3. Фаза побудови (Construction). Основна мета цієї фази – детальне прояснення вимог і розробка системи, що задовольняє їм, на основі спроектованої раніше архітектури. В результаті повинна вийти система, що реалізовує усі виділені варіанти використання. На цю фазу йде близько 50% часу і 65% трудомісткості одного циклу.

Результатами стадії розробки є:

- модель варіантів використання (завершена принаймні на 80%), що визначає функціональні вимоги до системи;
- перелік додаткових вимог, включаючи вимоги нефункціонального характеру;
- опис базової архітектури майбутньої системи, працюючий прототип; план розробки усього проекту, що відбиває ітерації і критерії оцінки для кожної ітерації.

4. Фаза впровадження (Transition). Мета цієї фази – зробити систему повністю доступною кінцевим користувачам. На цій стадії відбувається розгортання системи в її робочому середовищі, бета-тестування,

підгонка дрібних деталей під потреби користувачів. На цю фазу може йти близько 10% часу і 10% трудомісткості одного циклу.

Ключові ідеї RUP

Робочі продукти (артефакти), що виробляються в ході проекту, можуть бути представлені у вигляді баз даних і таблиць з інформацією різного типу, різних видів документів, початкового коду і об'єктних модулів, а також моделей, що складаються з окремих елементів.

Найбільш важливі з точки зору RUP артефакти проекту – це моделі, що описують різні аспекти майбутньої системи. Більшість моделей є наборами діаграм UML.

Перерахуємо основну техніку, використовувану в RUP:

1. Вироблення концепції проекту на його початку для чіткої постановки завдань.
2. Управління за планом.
3. Зниження ризиків і відстежування їх наслідків, як можна більше ранній початок робіт по подоланню ризиків.
4. Ретельне економічне обгрунтування усіх дій – робиться тільки те, що треба замовникові.
5. Як можна більше раннє формування базової архітектури.
6. Використання компонентної архітектури.
7. Прототипирование, інкрементна розробка і тестування.
8. Регулярні оцінки поточного стану.
9. Управління змінами, постійний відробіток змін ззовні проекту.
10. Націленість на створення продукту, працездатного в реальному оточенні.
11. Націленість на якість.
12. Адаптація процесу під потреби проекту.

Як видно, фази в цілому відповідають моделі водоспаду, з тим виключенням, що фаза супроводу не розглядається як окрема. В цілому, при розробці проекту RUP базується на чотирьох ключових ідеях.

1. Ключовою ідеєю процесу є його итеративность – кожна фаза, починаючи з розвитку, включає декілька ітерацій, на кожній з яких виконується свій шматочок аналізу, проектування, реалізації і тестування готового продукту (вірніше за його частину).

2. Увесь хід робіт спрямовується підсумковими цілями проекту (ітераціями), вираженими у вигляді ***прецедентів використання*** або ***варіантів використання (use cases)*** – сценаріїв взаємодії результуючої програмної системи з користувачами або іншими системами. Вимоги аналізуються також як і проект на кожній ітерації і в цілому дуже ретельно, але не до кінця – діє правило 70-80% – розробник повинен уявляти собі вимоги саме настільки, перш ніж починати кодування.

3. На етапі розвитку приймаються ключові рішення, що стосуються ***архітектури*** системи в цілому, її властивостей, функцій, використовуваних технологій і так далі. У кінці цієї фази реалізується 10-30% системи, але усі основні рішення вже прийняті (до 70-80%) і на етапі конструювання у рамках цих рішень система доводиться до кінця. Архітектура встановлює набір компонентів, з яких буде побудовано ПО, відповідальність кожного з компонентів (тобто вирішувані ним підзадачі у рамках загальних завдань системи), чітко визначає інтерфейси, через які вони можуть взаємодіяти, а також способи взаємодії компонентів один з одним.

4. Основою процесу розробки є ***плановані і керовані ітерації,*** об'єм яких визначається на основі архітектури (функціональність, що реалізовується у рамках ітерації, і набір компонентів).

RUP як продукт входить до складу інтегрованого комплексу інструментальних засобів **Rational Suite**, який існує в наступних варіантах:

- Rational Suite AnalystStudio – призначений для визначення і управління повним набором вимог до системи, що розробляється;

- Rational Suite DevelopmentStudio – призначений для проектування і реалізації ПЗ;
- Rational Suite TestStudio – є набір продуктів, призначених для автоматичного тестування додатків;
- Rational Suite Enterprise – забезпечує підтримку повного життєвого циклу

ПЗ і призначений як для менеджерів проекту, так і окремих розробників, що виконують декілька функціональних ролей в команді розробників.

До складу Rational Suite, окрім самої технології RUP як продукту, входять наступні компоненти:

- Rational Rose – засіб візуального моделювання (аналізу і проектування), що використовує мову UML;
- Rational XDE – засіб аналізу і проектування, інтегрований з платформами MS Visual Studio .NET і IBM WebSphere Studio Application Developer;
- Rational Requisite Pro – засіб управління вимогами, призначене для організації спільної роботи групи розробників;
- Rational Rapid Developer – засіб швидкої розробки додатків на платформі Java 2 Enterprise Edition;
- Rational SoDA – засіб автоматичної генерації проектної документації;
- Rational Quantify – засіб кількісного визначення вузьких місць, що впливають на загальну ефективність роботи програми;
- Rational TestManager – засіб планування функціонального і навантаження тестування;
- Rational TestFactory – засіб тестування надійності;
- Rational Quality Architect – засіб генерації коду для тестування.

Засоби автоматичної генерації коду, використовуючи інформацію, що міститься в діаграмах класів і компонентів, формують файли описів класів. Створюваний таким чином скелет програми може бути уточнений шляхом

прямого програмування на відповідній мові (основні мови, підтримувані Rational Rose, – C++ і Java).

Уніфікований процес RUP хоча і є досить складною ітеративною моделлю розробки життєвого циклу ПЗ, все ж може з успіхом застосуються, якщо на деяких етапах (фазах) допускати деякі «вільності».

Серед проблем власне кодування, слід виділити небажання деяких розробників (а також деяких замовників) використати компоненти сторонніх виробників. Щоб не винаходити велосипед, при написанні коду також можуть використовуватися готові програмні конструкції (готові рішення), типові рішення, шаблони, так звані «**патерни» (**design patterns – зразки проектування або просто patterns).

Проте які б поліпшення ми не вносили, техніку RUP властиві серйозні обмеження і недоліки:

1. Він уніфікований, тобто підходить для різнорідних процесів, проектів, розробок, що робить його трохи заплутаним і неконкретним (мало конкретних чітких рекомендацій).
2. Він важкуватий для невеликих проектів і колективів розробників, особливо коли бюджет і терміни проектів невеликі.

Він вимагає глибокого осмислення вимог, як і традиційний водоспад (хоча і залишає на потім 30%). У реальних ситуаціях і 70% відразу отримати важко!

Останнім часом все більшу популярність стали набирати так звані «**гнучкі» (*«живі»)*** методи розробки ПЗ (Agile Software Development). Серед них найпоширенішим являться ***Екстремальне програмування (eXtreme Programming, XP)*** – полегшений (рухливий) процес (чи методологія), головний автор якого – Кент Бек (1999) [5].

XP-процес орієнтований на групи малого і середнього розміру, що будують програмне забезпечення в умовах невизначених або швидко таких, що змінюються вимог.

Основні принципи «живої» розробки ПЗ зафіксовані в маніфесті «живої» розробки, що з'явився в 2000 році:

1. Люди, що беруть участь в проекті, і їх спілкування важливіші, ніж процеси і інструменти.
2. Працююча програма важливіша, ніж вичерпна документація.
3. Співпраця із замовником важливіша, ніж обговорення деталей контракту.
4. Відробіток змін важливіший, ніж наслідування планів.

«Живі» методи з'явилися як протест проти надмірної бюрократизації розробки ПЗ, великої кількості побічних що не є необхідними для отримання кінцевого результату документів, які доводиться оформляти при проведенні проекту відповідно до більшості «важких» процесів. Велика частина таких робіт і документів не має прямого відношення до розробки ПЗ і забезпечення його якості, а призначена для дотримання формальних пунктів контрактів на розробку, отримання і підтвердження сертифікатів на відповідність різним стандартам.

Основна ідея XP-процесу – усунути високу вартість зміни, характерну для додатків з використанням об'єктів, **патернів** (рішення типових проблем в певному контексті або готові програмні конструкції) і реляційних баз даних. Тому XP-процес має бути високо динамічним процесом. XP-група має справу зі змінами вимог на всьому протязі ітераційного циклу розробки, причому цикл складається з дуже коротких ітерацій.

За твердженням авторів XP, ця методика є не стільки наслідуванням якихось загальних схем дій, скільки застосування комбінації відповідної техніки (практик). Кожна техніка важлива, і без її використання розробка вважається такою, що йде не по XP.

1. **Гра в планування (Planning game) –** швидке визначення зони дії наступної реалізації шляхом об'єднання ділових пріоритетів і технічних оцінок. Замовник формує зону дії, пріоритетність і терміни з точки зору бізнесу, а розробники оцінюють і простежують просування (прогрес).

2. **Часта зміна версій (Small releases) –** швидкий запуск у виробництво простої системи, тобто найперша працююча версія повинна з'явитися якнайшвидше, і тут же повинна почати використовуватися. Для реалізації нових версій вводяться ще жорсткіші обмеження на тривалість однієї ітерації.

3. **Метафора (Metaphor) –** уся розробка проводиться на основі простої, загальнодоступної історії про те, як працює уся система. Метафора в досить простому і зрозумілому команді виді повинна описувати основний механізм роботи системи. Це поняття нагадує архітектуру, але повинне набагато простіше, усього у вигляді однієї-двох фраз описувати основну суть прийнятих технічних рішень.

4. **Просте проектування (Simple design)** – проектування виконується настільки просто, наскільки це можливо в даний момент. Не потрібно додавати функції заздалегідь – тільки після явного прохання про це. Уся зайва складність віддаляється, як тільки виявляється.

5. **Тестування (Testing) –** безперервне написання тестів для модулів, які повинні виконуватися бездоганно. Розробники спочатку пишуть тести, потім намагаються реалізувати свої модулі так, щоб тести спрацьовували.

6. **Реорганізація (рефакторинг, Refactoring)** – наведення ладу в коді, переробка окремих файлів, видалення максимальної кількості незрозумілих фрагментів коду, об'єднання класів на основі їх схожої функціональності, корекція коментарів, осмислене перейменування об'єктів. Система реструктурується у зв'язку з додаванням нової функціональності, але її поведінка не змінюється; мета – усунути дублювання, спростити систему.

7. **Парне програмування (Pair programming) –** увесь код пишеться двома програмістами, працюючими на одному комп'ютері. Об'єднання в пари довільно і міняється від завдання до завдання. Той, в чиїх руках клавіатура, намагається найкращим способом вирішити поточне завдання. Другий програміст аналізує роботу першого і дає поради, обмірковує наслідки тих або інших рішень, нові тести, менш прямі, але гнучкіші рішення.

8. **Колективне володіння кодом (Collective ownership) –** будь-який розробник може покращувати будь-який код системи у будь-який час. Ніхто не повинен виділяти свою власну область відповідальності, уся команда в цілому відповідає за увесь код.

9. **Безперервна інтеграція (Continuous integration) –** система інтегрується і будується багато разів в день, у міру завершення кожного завдання. Безперервне регресійне тестування, тобто повторення попередніх тестів, гарантує, що зміни вимог не приведуть до регресу функціональності.

10. **40-годинний тиждень (40 – hour week) –** як правило, працюють не більше 40 годин в тиждень. Не можна подвоювати робочий тиждень за рахунок наднормових робіт.

11. **Локальний замовник (On – site customer) –** в групі увесь час повинен знаходитися представник замовника, дійсно готовий відповідати на питання розробників. Його обов'язком є досить оперативні відповіді на питання будь-якого типу, що стосуються функцій системи, її інтерфейсу, необхідної продуктивності, правильної роботи системи в складних ситуаціях.

12. **Стандарти (стилі) кодування (Coding standards) –** повинні витримуватися правила, що забезпечують однакове представлення програмного коду в усіх частинах програмної системи. Код розглядається як найважливіший засіб спілкування усередині команди. Ясність коду – один з основних пріоритетів.

13. **Відкритий робочий простір (Open workspace)**. Команда розміщується в одному, досить просторому приміщенні, для спрощення комунікації і можливості проведення колективних обговорень при плануванні і ухваленні важливих технічних рішень.

14. **Зміна правил з потреби (just rules).** Кожен член команди повинен прийняти перераховані правила, але при виникненні необхідності команда може поміняти їх, якщо усі її члени прийшли до згоди з приводу цієї зміни.

Недоліками цього підходу деякі фахівці вважають нездійсненність в такому стилі досить великих і складних проектів, неможливість планувати терміни і трудомісткість проекту на досить довгу перспективу і чітко передбачити результати тривалого проекту в термінах співвідношення якості результату і витрат часу і ресурсів.

Scrum методологія

Перевагами XP є велика гнучкість і простота розуміння, проте в повному об'ємі XP не була використана навіть її авторами. Крім того, відомі і успішно впроваджуються окремі практики XP, як, наприклад, парне програмування, колективне володіння кодом, і, звичайно ж, рефакторинг кода. Ідея простого, ненадмірного дизайну проекту також зробила значний вплив на світ розробників ПО.

Більше практичним «гнучким» методом розробки є Scrum [35].

Історія

У 1986 японські фахівці Hirotaka Takeuchi і Ikujiro Nonaka опублікували повідомлення про новий підхід до розробки нових сервісів і продуктів (не обов'язково програмних). Основу підходу складала згуртована робота невеликої універсальної команди, яка розробляє проект на усіх фазах. Наводилася аналогія з регбі, де уся команда рухається до воріт супротивника як єдине ціле, передаючи (пасуючи) м'яч своїм гравцям як вперед, так і назад. На початку 90-х років цей підхід став застосовуватися в програмній індустрії і набув назви **Scum** (термін з регбі, що означає, – сутичка), в 1995 році Jeff Sutherland і Ken Schwaber представили опис цього підходу на OOPSLA '95 – одній з найавторитетніших конференцій в області програмування. Відтоді метод активно використовується в індустрії і багаторазово описаний в літературі.

Загальний опис

Метод Scrum дозволяє гнучко розробляти проекти невеликими командами (7 чоловік плюс/мінус 2) в ситуації вимог, що змінюються. При цьому процес розробки итеративен і надає велику свободу команді.

Спочатку створюються вимоги до усього продукту. Потім з них вибираються найактуальніші і створюється план на наступну ітерацію. Впродовж ітерації плани не міняються (цим досягається відносна стабільність розробки), а сама ітерація триває 2-4 тижні. Вона закінчується створенням працездатної версії продукту (робочий продукт), яку можна пред'явити замовникові, запустити і продемонструвати, хай і з мінімальними функціональними можливостями. Після цього результати обговорюються і вимоги до продукту коригуються.

Ролі

У Scrum є всього три види ролей.

Власник продукту (Product Owner) – це менеджер проекту, який представляє в проекті інтереси замовника. У його обов'язки входить розробка початкових вимог до продукту (Product Backlog), своєчасна їх зміна вимог, призначення пріоритетів, дат постачання і ін. Важливо, що він абсолютно не бере участь у виконанні самої ітерації.

Scrum-мастера (Scrum Master) забезпечує максимальну працездатність і продуктивну роботу команди – як виконання Scrum-процесса, так і рішення господарських і адміністративних завдань. Зокрема, його завданням є обгороджування команди від усіх дій ззовні під час ітерації.

Scrum-команда (Scrum Team) – група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Першим завданням команди є постановка для ітерації реально досяжних і пріоритетних для проекту в цілому завдань (на основі Project Backlog і при активній участі власника продукту і Scum-мастера). Другим завданням є виконання цього завдання щоб то не було, у відведені терміни і із заявленою якістю.

Практики

У Scrum визначені наступні практики.

Sprint Planning Meeting. Проводиться на початку кожного Sprint. Спочатку Product Owner, Scurm-мастер, команда, а також представники замовника і інші зацікавлені особи визначають, які вимоги з Project Backlog найбільш пріоритетні і їх слід реалізовувати у рамках цього Sprint. Формується

Sprint Backlog. Далі Scurm-мастер і Scrum-команда визначають те, як саме буде досягнута певна вище мета з Sprint Backlog. Для кожного елементу Sprint Backlog визначається список завдань і оцінюється їх трудомісткість.

Daily Scrum Meeting – п'ятнадцятихвилинна щоденна нарада, метою якої є досягти розуміння того, що сталося з часу попередньої наради, скоректувати робочий план згідно реаліям сьогоднішнього дня і позначити шляхи рішення існуючих проблем. Кожен учасник Scrum-команди відповідає на три питання: що я зробив з часу попередньої зустрічі, мої проблеми, що я робитиму до наступної зустрічі?

У цій нараді (15-20 хвилин) може брати участь будь-яка зацікавлена особа, але тільки учасники Scrum-команди мають право приймати рішення. На них лежить відповідальність за їх власні слова, і, якщо хтось з боку втручається і приймає рішення за них, тим самим він знімає відповідальність за результат з учасників команди.

Sprint Review Meeting. Проводиться у кінці кожного Sprint. Спочатку Scrum-команда демонструє Product Owner зроблену впродовж Sprint роботу, а той у свою чергу веде цю частину мітингу і може запросити до участі усіх зацікавлених представників замовника. Product Owner визначає, які вимоги з Sprint Backlog були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в Sprint Backlog для наступної ітерації. У другій частині мітингу робиться аналіз минулого спринту, який веде Scrum-мастер. Scrumкоманда аналізує в останньому Sprint позитивні і негативні моменти спільної роботи, робить висновки і приймає важливі для подальшої роботи рішення.

Хід роботи

Завдання 1. За допомогою пошуку в Інтернеті знайдіть інформацію про сучасні методології управління ІТ-проектами. Подайте підстави для їх класифікації. Для кожної основи наведіть приклади методологій.

Завдання 2. З отриманого списку важковагових методологій управління ІТ-проектами виберіть одну. Проведіть дослідження методології. Результат подайте в таблиці (таблиця 2).

Завдання 3. З отриманого списку легких методологій управління ІТ-проектами виберіть одну. Проведіть дослідження методології. Результат подайте в таблиці (таблиця 2).

Завдання 4. Виберіть будь-яку із проаналізованих методологій. Створіть про неї презентацію на 10-15 слайдів. Виступіть у групі, будьте готові відповісти на запитання.

Таблиця 2 - Особливості методики

Характеристика	Опис
Повна назва методології	
Автори	
Історія виникнення	
Країна появи	
Основні засади, підходи	
Програмні засоби реалізації методології	
Чи використовується в даний час	
Приклади успішних проектів, реалізованих за допомогою даної методології	

Контрольні питання

1. Що таке методологія управління ІТ-проектом?
2. Які види методологій ви знаєте?
3. У чому особливості великовагових та легких методологій управління?
4. Наведіть приклади методологій для розробки ІТ-проектів.