Додатковий матеріал до Лекції №1

Пошук компромісу між складними формальними процесами і полегшеними методами швидкої розробки призвів до появи методології розробки під назвою agile ("моторний", "швидкий", "живий"). Принципи цього підходу сформульовані в Agile Manifesto (www.agilemanifesto.org), який був написаний в лютому 2001 року сімнадцятьма представниками ряду "нетрадиційних" напрямів в програмуванні, включаючи **XP [1], Feature Driven Development, Crystal, Adaptive Software Development, SCRUM**.

У новому підході на перший план виходять креативні можливості яскравих індивідуальностей в колективі програмістів. При цьому стимулюється максимальне використання потенціалу спільної розробки у поєднанні з мінімумом обмежень, що накладаються формальними методами і процесами. Постійній взаємодії із замовниками віддається перевага над строгим наслідуванням формальних специфікацій, і робота організовується так, щоб швидко реагувати на зміни у вимогах, а не пунктуально реалізовувати спочатку заданий план.

Процес створення програмного забезпечення по принципах **agile** повинен мати ряд ключових характеристик [2].

- *Модульність*. Процес розробки розбивається на окремі завдання, які в сукупності дозволяють перетворити концепцію системи в реалізацію. Завдання - одиниця роботи, що має певну мету.
- *Ітеративність*. Процес розробки розбивається на короткі за часом цикли. В ході кожної ітерації виконується деяка множина завдань, ітерації завершуються впродовж декількох тижнів. Ітеративність відповідає тим реаліям розробки, що в системі неможливо відразу задовольнити усі вимоги замовника, доводиться постійно вносити зміни, щоб усунути помилки і точніше наслідувати поставлені цілі. Тому ітеративність має на увазі безперервну взаємодію із замовником.
- *Обмеженість в часі*. Для кожної ітерації задаються жорсткі обмеження за часом, і відповідно до цього складається календарний план проекту.
- *Економічність*. У процес розробки включається мінімальне число завдань, необхідних для досягнення поставленої мети і зниження ризику проекту.
- *Адаптивність*. В ході розробки можуть виникнути нові ризики, не враховані при плануванні проекту, і може виявитися так, що раніше визначені завдання не дозволяють реалізувати проект. В цьому випадку список завдань, що становлять процес, може бути змінений.
- *Поступовість*. Проект розбивається на частини, які можуть розроблятися паралельно, в різний час і з різною швидкістю. Модульне тестування для різних частин проекту проводиться незалежно, після його завершення частина інтегрується в систему.
- *Орієнтація на людей*. Для реалізації кожної частини проекту формуються невеликі групи розробників, що дозволяє їм не почувати себе ізольовано у великому колективі проекту.
- *Колаборативність*. Стимулюється постійна співпраця між учасниками проекту. Це необхідно для того, щоб ефективно інтегрувати різні компоненти системи при паралельній роботі.
- *Взаємодоповнюваність*. Деякі завдання, що реалізуються в сукупності, дають кращий результат в порівнянні з реалізацією ізольовано один від одного. Певні завдання можна використовувати для оцінки і поліпшення результатів раніше виконаних завдань.

Екстремальне програмування

Екстремальне програмування (Extreme Programming, XP) виникло як еволюційний метод розробки ПО "знизу-вгору". Цей підхід є прикладом так званого методу "живої" розробки (Agile Development Method). До групи "живих" входять, окрім екстремального програмування, методи SCRUM, DSDM (Dynamic Systems Development Method, Метод розробки динамічних систем), Feature - Driven Development (Розробка, керована характеристиками результату) і ін.

Основні принципи "живої" розробки ПО зафіксовані в маніфесті "живої" розробки [3].

- Люди їх спілкування важливіші, ніж процеси і інструменти
- Працююча програма важливіша, ніж вичерпна документація
- Співпраця із замовником важливіша, ніж обговорення деталей контракту
- Відпрацювання змін важливіша, ніж дотримування планів

Проте, XP має свою схему процесу розробки (хоча широко використовуване розуміння "процесу розробки" суперечить "живості" розробки).

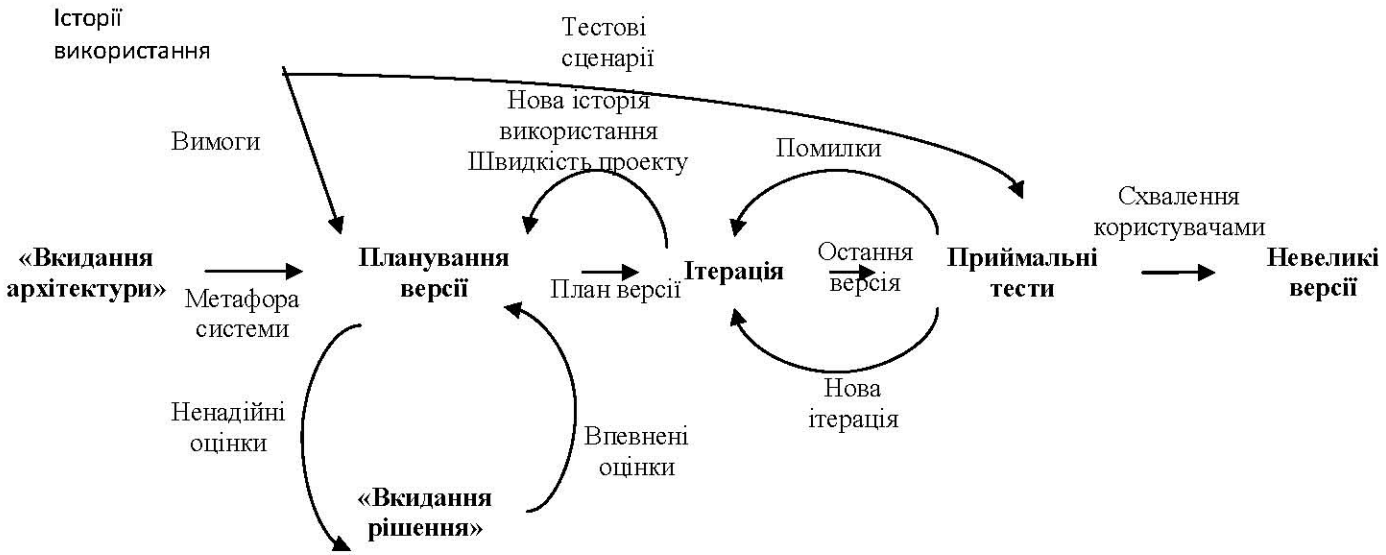

Схема процесу XP

За твердженням авторів XP, ця методика є використанням комбінації наступної техніки (при цьому кожна техніка важлива, і без її використання розробка вважається такою, що йде не по XP).

- *Живе планування (planning game)*

Його завдання якнайшвидше визначити об'єм робіт, який треба зробити до наступної версії ПО. Рішення приймається на основі, в першу чергу, бізнес-пріоритетів замовника і, по-друге, технічних оцінок. Плани змінюються, як тільки вони починають розходитися з дійсністю або побажаннями замовника.

- *Часта зміна версій (small releases)*

Найперша працююча версія повинна з'явитися якнайшвидше, і відразу ж повинна почати використовуватися. Наступні версії готуються через досить короткі проміжки часу.

- *Метафора (metaphor) системи*

Метафора в досить простому і зрозумілому команді виді повинна описувати основний механізм роботи системи.

- *Прості проектні рішення (simple design)*

У кожен момент часу система має бути сконструйована так просто, наскільки це можливо. Не потрібно додавати функції "заздалегідь" - тільки після ясного прохання про це. Уся зайва складність віддаляється, як тільки виявляється.

- *Розробка на основі тестування (test - driven development)*

Розробники спочатку пишуть тести, потім намагаються реалізувати свої модулі так, щоб тести спрацьовували. Замовники заздалегідь пишуть тести, що демонструють основні можливості системи, щоб можна було побачити, що система дійсно заробила.

- *Постійна переробка (refactoring)*

Програмісти постійно переробляють систему для усунення зайвої складності, збільшення зрозумілості коду, підвищення його гнучкості, але не змінюючи його поведінки, що перевіряється прогоном тестів. При цьому перевага віддається елегантнішим і гнучкішим рішенням, в порівнянні з тими, що просто дають потрібний результат.

- *- Програмування парами (pair programming)*

Увесь код пишеться двома програмістами на одному комп'ютері. Об'єднання в пари довільне і змінюється від завдання до завдання. Той, в чиїх руках клавіатура, намагається найкращим способом вирішити поточне завдання. Другий програміст обмірковує наслідки, нові тести, менш прямі, але гнучкіші можливості рішення.

- *Колективне володіння кодом (collective ownership)*

У будь-який момент будь-який член команди може змінити будь-яку частину коду, але він же несе відповідальність за ці зміни.

- *Постійна інтеграція (continuous integration)*

Система збирається і проходить інтеграційне тестування якомога частіше, по кілька разів в день, кожного разу, коли пара програмістів закінчує реалізацію чергової функції.

- *40-годинний робочий тиждень*

Наднормова робота розглядається як ознака великих проблем в проекті. Не допускається наднормова робота 2 тижні підряд - це виснажує програмістів і робить їх роботу значно менш продуктивною.

- *Включення замовника в команду (on - site customer)*

У складі команди розробників постійно знаходиться представник замовника, який доступний впродовж усього робочого дня і здатний відповідати на усі питання про систему.

- *Використання коду як засобу комунікації*

Код розглядається як найважливіший засіб спілкування усередині команди. Ясність коду - один з основних пріоритетів. Наслідування стандартів кодування, що забезпечують таку ясність, обов'язкове. Такі стандарти, окрім ясності коду, повинні забезпечувати мінімальність виразів (заборона на дублювання коду і інформації) і мають бути прийняті усіма членами команди.

- *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.

- *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.

Як бачимо, XP розраховане на використання у рамках невеликих команд (не більше 10 програмістів), що підкреслюється і авторами цієї методики, більший розмір команди руйнує легкість комунікації.

Adaptive Software Development

Adaptive Software Development (ASD) - одна з нових методологій, які з'явилися як альтернатива традиційним, орієнтованим на процес, методам управління розробкою ПО. ASD, Extreme Programming (XP), Lean Development, SCRUM і сімейство методологій Crystal, звичайно, багато в чому відрізняються один від одного, проте у них усіх є одна загальна риса - в основу в них покладений людський чинник, результати роботи і мінімізація самого процесу при максимальному збільшенні взаємодії між людьми. Усі ці методології були розроблені виходячи з об'єктивних реалій сучасного високотехнологічного бізнесу, який відрізняється величезною швидкістю розвитку і високою мінливістю.

Чим би ви не керували - тестуванням, командою розробників або усім проектом в цілому, прийшов час переглянути ті цінності і положення, які лежать в основі процесу керівництва.

Практики ASD базуються на принципі безперервної адаптації, завдяки якій виникає інша філософія і інший життєвий цикл проекту, коли постійні зміни стають нормою.

Адаптивна розробка програмного забезпечення - визначувана місією, заснована на компонентах, використовує ітеративні цикли і цикли з відомою тривалістю, визначувані мірою ризику, допускає зміни.

У ASD звичайний статичний життєвий цикл Plan - Design - Build (Планування - Проектування - Конструювання) замінений на динамічний - Speculate - Collaborate - Learn (Обдумування - Взаємодія - Навчання).

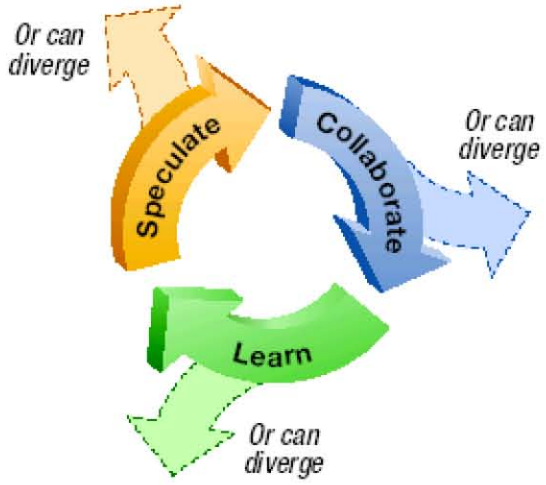

FIGURE 1 The adaptive lifecycle

Цей цикл ставить своєю метою безперервне навчання. Він пов'язаний з постійними змінами, повторними оцінками, спробами передбачити невідоме на даний момент майбутнє проекту і вимагає тісної взаємодії між розробниками, тестувальниками і замовниками. (Зверніть увагу, що увесь цей цикл не завжди є правильним колом. Навіть при ітеративному процесі можна іноді відхилятися убік, щоб вивчити недосліджені досі області).

Методологія ASD побудована на концептуальній базі теорії складних адаптивних систем. Вона розрахована на використання в екстремальних проектах, в яких переважають швидкий темп розробок, непередбачуваність і часті зміни. Є проекти, які не можуть вважатися екстремальними, проте для усіх інших ASD підходить набагато краще, ніж будь-який традиційний підхід до розробки ПО.

Didkovska Maryna

Dynamic System Development Method (DSDM)

Усе починається з вивчення здійсненності програми і сфери її застосування. У першому випадку, ви намагаєтеся зрозуміти, чи підходить DSDM для цього проекту. Вивчати сферу застосування програми передбачається на короткій серії семінарів, де програмісти дізнаються про ту сферу бізнесу, для якої їм належить працювати. Тут же обговорюються основні положення, що стосуються архітектури майбутньої системи і план проекту.

Далі процес ділиться на три взаємозв'язані цикли:

- цикл функціональної моделі відповідає за створення аналітичної документації і прототипів,
- цикл проектування і конструювання - за приведення системи в робочий стан,
- цикл реалізації - забезпечує розгортання програмної системи.

Базові принципи, на яких будується DSDM, це активна взаємодія з користувачами, часті випуски версій, самостійність розробників в ухваленні рішень і тестування впродовж усього циклу робіт. Як і більшість інших гнучких методологій, DSDM використовує короткі ітерації, тривалістю від двох до шести тижнів кожна. Особлива увага приділяється високій якості роботи і адаптуванні до змін у вимогах.
Її інфраструктура має усі риси, характерні для більш зрілих традиційних методологій, тоді як основні принципи відповідають новим гнучким методологіям.

Feature Driven Development (FDD)

Feature Driven Development (FDD) - функціонально-орієнтована розробка.
Як і інші адаптивні методології, вона концентрується на коротких ітераціях, кожна з яких служить для опрацювання певної частини функціональності системи. Згідно FDD, одна ітерація триває два тижні.

FDD включає п'ять процесів, перші три з них відносяться до початку проекту, останні два повторюються для кожної функції:

- Розробка загальної моделі
- Складання списку необхідних функцій системи
- Планування роботи над кожною функцією
- Проектування функції
- Конструювання функції

Останні два кроки необхідно робити під час кожної ітерації. При цьому кожен процес розбивається на завдання і має критерії верифікації.

Розробники в FDD діляться на "хазяїв класів"("class owners") і "головних програмістів" ("chief programmers"). Голвні програмісти - це найбільш досвідчені розробники. Саме їм доручається розробка конкретних властивостей системи. Проте вони не займаються цим самостійно: старший програміст визначає, які класи зайняті в реалізації цієї конкретної властивості, після чого збирає команду з власників необхідних класів, яка і займатиметься розробкою. Сам він діє як координатор, головний проектувальник і керівник, а на долю власників класів залишається, здебільшого, безпосереднє кодування. Головні програмісти залучають хазяїв задіяних класів до роботи над черговою властивістю. Для порівняння, в XP немає персонально відповідальних за класи або методи. До роботи над будь-яким класом може притягуватися будь-який з учасників розробки.

Робота над проектом припускає часті зборки і ділиться на ітерації, кожна з яких припускає реалізацію певного набору функцій.

На жаль, такі області, як забезпечення якості, оцінка ризиків, ця модель не враховує.

FDD підходить для невеликих проектів, термін реалізації яких складає декілька місяців. Для довгострокових проектів, з терміном виробництва рік або більше, необхідно вибрати іншу, більш формалізовану модель.

Адаптивна модель життєвого циклу Scrum

Основною передумовою виникнення методології Scrum була неможливість використання існуючих на той момент моделей (каскадна модель, RUP) для динамічного адаптування програмного продукту до нових вимог.

Модель Scrum включає три стадії :

- Підготовка (pregame phase)
- Розробка запланованих функціональних можливостей (development phase)
- Завершення (postgame phase)

Структура моделі Scrum зображена на рис.1.6

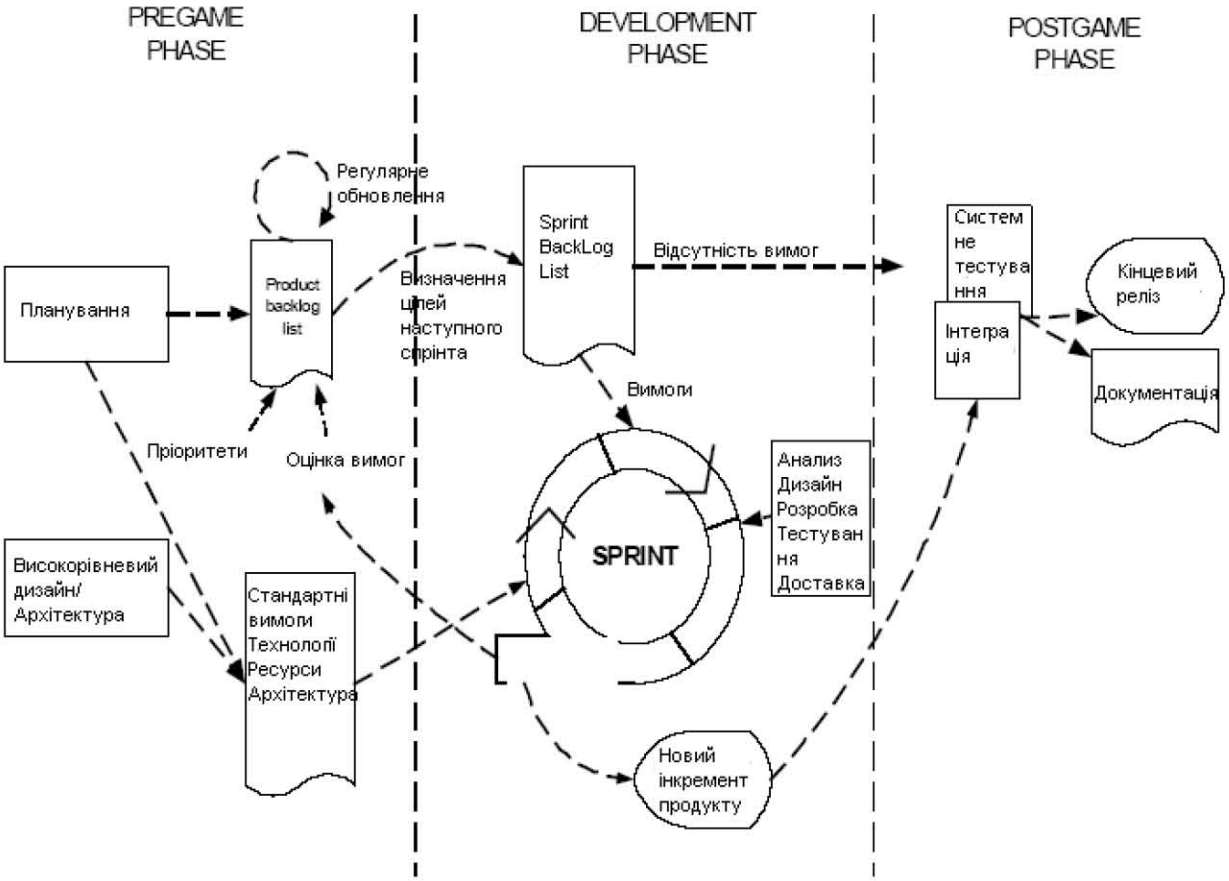

Рис.1.6 Адаптивна модель життєвого циклу Scrum

Розглянемо детально стадії моделі Scrum

Стадія підготовки ітерації включає в себе дві під стадії:

1) планування

- Розробка повного Product Log.

- Визначення дати кінцевого релізу та функціоналу для однієї або декількох ітерації
- Вибір початкової ітерації
- Визначення команди розробників для ітерації
- Визначення можливих ризиків та засобів їх контролю
- Перегляд та можливе коректування Product Log
- Вибір та валідація інструментів та технології для розробки
- Визначення бюджету релізу, включаючи розробки, маркетинг, тренінги та інше
- Узгодження з керівництвом бюджету та функціонального вмісту релізу

2) високорівневий дизайн

- перегляд зазначених в Product Log специфікації
- визначення необхідних змін в специфікаціях для імплементації нових функціональних можливостей програмного продукту
- проведення аналізу систему для виділення необхідних дій для створення, розширення чи обновлення існуючого програмного забезпечення, так щоб воно задовольняло поставленим умовам
- уточнення архітектури системи для підтримання запланованих в Product Log специфікації
- виявлення проблем та питань, що можуть виникнути в процесі розробки
- створення плану імплементації для кожної специфікації Product Log, обговорення та затвердження цих планів командою розробників

Фаза розробки - це ітеративний процес розробки запланованих функціональних можливостей. Керівництво проекту визначає час, функціональне наповнення, якість кожної ітерації. По завершенню кожної ітерації відбувається аналіз отриманих результатів. Розробка включає в себе наступні процеси :

- періодичне обговорення отриманих результатів та планів наступного релізу
- перегляд та корегування стандартів, яким буде відповідати розроблений продукт
- ітеративні спринти, поки програмних продукт не буде готовий до випуску

Спринт (sprint) – це сукупність усіх завдань, що будуть виконувати розробники протягом визначеного періоду, за звичай від одного до чотирьох тижнів. Довжина спринту залежить від складності продукту, оцінених ризиків. Кожен спринт включає всі традиційні фази розробки програмного забезпечення: специфікації, аналіз, дизайн, розробка, тестування та демонстрація розроблених функціональних можливостей. Довжина спрінту має бути фіксованою, встановлюється за вибором команди розробників. В теорії Scrum довжина спринту – 30 днів.

Стадія завершення в Scrum містить усі необхідні завдання для остаточного випуску системи, після того як закінчилася розробка усіх запланованих функціональних можливостей. Це такі процеси, як системне тестування. написання документації, підготовка необхідних тренінгів тощо.

Розглянемо організаційні аспекти для ефективного впровадження та використання моделі Scrum

1) Команда розробників має бути невеликою та крос-функціональною (дозволяє уникнути ризиків пов'язаних з незамінністю членів команди);
2) Команда має бути самоорганізованою та має вирішувати проблеми на кожному спринті самотужки;

3) З командою розробників постійно працює спеціаліст із технології Scrum – Scrum Master, який не відповідає за технічні аспекти розробки програмного продукту, але забезпечує дотримання основних принципів Scrum.

4) Робота організовується через Product log та Sprint BackLog. Product log – список основних вимог до програмного продукту, які мають бути реалізовані на протязі проекту. Sprint Backlog – це список функціональних можливостей, що будуть реалізовані протягом поточного спринту. Наповнення Sprint Backlog відбувається на стадії планування ітерації-спринту

5) Інтереси замовника в команді представляє Product Owner. Його функції полягають в наповненні, управлінні та контролю Product Log , оцінці часу для реалізації історій з Product Log, прийняття остаточне рішення щодо реалізації історій з Product Log, написанням специфікації для затверджених до імплементації нових функціональних можливостей.

6) Використання спеціальних засобів для покращення комунікації, контролю над станом проекті, плануванні та огляді ітерацій. Це такі засоби, як daily scrum meetings (щоденні зустрічі команди розробників для контролю перебігу проекту), sprint planning meetings (планування ітерацій scrum командою), sprint retrospective meetings(аналіз результатів спринтів та проекту в цілому).

Література

1. T. Murphy, Accelerating J2EE Application Delivery. Meta Group White Paper, June 2003.
2. IEEE Standard Glossary of Software Engineering Terminology, 1997.
3. T. Murphy, Mastering the requirements of requirements management. Meta Group, April 2003.