**Лекція 7. Методологія розробки програмного забезпечення**

**Методологія розробки програмного забезпечення** (англ. *Software development methodology*) — сукупність методів, застосовуваних на різних стадіях життєвого циклу розробки програмного забезпечення, що мають спільний філософський підхід та, відповідно до цього підходу, дозволяють забезпечити найкращу ефективність процесів розробки.

Кожна методологія характеризується:

* Філософським підходом або основними принципами. Ці принципи, від яких залежить ефективність всієї методології, зазвичай можна коротко сформулювати і легко пояснити;
* Узгодженою множиною моделей та методів, які реалізують дану методологію;
* Концепціями (поняттями), що дозволяють більш точно визначити методи.

Методологія розробки ПЗ - це сукупність принципів, система ідей, понять, способів, методів і засобів, які в кінцевому підсумку визначатимуть стиль розробки ПЗ. Саме вона визначає, ЯК буде виконуватися розробка. Зазвичай це – реалізація якого-небудь певного стандарту. Вибір конкретної методології залежить від розміру команди, від специфіки і складності проекту, від стабільності і зрілості процесів у компанії, від особистих уподобань, в кінці кінців. Методології являють собою ядро теорії управління розробкою ПЗ.

Раніше до методології застосовувалася класифікація, що їх на два типи: ітераційні та водоспадні (виходячи з застосовуваних моделей життєвого циклу). Сучасна загальна класифікація поділяє також на дві групи: прогнозовані і адаптивні.

Прогнозовані методології, орієнтовані на детальне планування майбутнього. Завдання і ресурси відомі на всьому протязі терміну проекту. Несподівані зміни в ресурсах, завданнях та вимогах може призвести до зміни всього плану, дизайну проекту.

Адаптивна розробка замінює водоспадну модель повторюваними серіями обдумування, співробітництва та навчання. Цей динамічний цикл передбачає постійне навчання та адаптацію до виникаючих станів проекту. Характеристиками життєвого циклу АРП є зосередженість уваги, ітеративність, обмеженість за часом, управління ризиком, терпимість до змін.

### Класифікація по ядрам

При зверненні до методології,  яка має ядро (англ. *core*), відповідне способу опису алгоритму, та додаткові особливості, можна виділити такі п'ять основних ядер методологій:

* Методологія імперативного програмування
* Методологія ООП
* Методологія функціонального програмування
* Методологія логічного програмування
* Методологія програмування в обмеженнях

Можна помітити, що ці методології знаходяться на шкалі від навігаційних (покрокове управління виконанням) до специфічних (визначення вимог до результату).

### Класифікація за топологічною специфікою

Специфіка (топологічна специфіка) — спосіб вибору методів для уточнення ядра методології. Критерієм якості тієї чи іншої топології можуть бути загальні витрати на розробку ПЗ. У свою чергу, витрати на розробку залежать серед іншого від ключових мовних абстракцій: абстракції даних, управління і модульності. Наприклад, в імперативній методології можна дотримуватися методів структурного програмування, що дає більш вигідну топологію з погляду мовних засобів. Результатом є методологія структурного програмування.

### Класифікація за специфікою реалізації

У відповідності до архітектури апаратного забезпечення, реалізація може бути централізованою або паралельною. Наприклад, методологія (імперативного) паралельного програмування, методологія логічного паралельного програмування.

Крім того, методологія може бути гібридної. Наприклад, найбільш часта суміш функціонального та логічного програмування.

**Прогнозовані методології**

Waterfall Model - "водоспад" або каскадна модель. Одна з найстаріших методологій. Передбачає послідовне проходження етапів розробки ПЗ, кожен з яких повинен закінчиться до того, як почнеться наступний. . Вартість і строки розробки заздалегідь визначені, завдяки такій жорсткості, за даної методології досить легко керувати проектом чому роботи проходять швидко. Каскадна модель дає відмінний результат тільки в проектах з заздалегідь чітко заданими вимогами, методиками їх реалізації. Оскільки тестування починається тільки після повного завершення етапу, то виправлення помилок може виявитися дорогим, оскільки для початку виправлень доводиться чекати закінчення розробки.

Методологію "водоспад"рекомендують використовувати, коли вимоги до проекту відомі, зрозумілі і фіксовані. Будь-яких суперечностей між ними немає. Немає проблем до залучення програмістів необхідної кваліфікації. Проект відносно невеликий.

### В число найбільших переваг методики Waterfall увійшли:

* *зрозуміла і проста структура процесу розробки*: це знижує поріг входження для команд;
* *зручна звітність*: можна легко відстежити ресурси, ризики, витрачений час і фінанси завдяки суворій етапності процесу розробки та детальної документації проекту
* *стабільність задач*: задачі, які стоять перед продуктом, ясні команді з самого початку розробки, і залишаються незмінними протягом усього процесу
* *оцінка вартості та термінів здачі проекту*: терміни випуску готового продукту, як і його підсумкова вартість можуть бути прораховані до моменту запуску розробки.

### Серед недоліків водоспадного методу можна виділити:

* *позбавлений гнучкості процес*: так, якщо проект вимагає більше тимчасових і фінансових ресурсів, чим можливо, то під ніж піде фаза тестування. Згідно з дослідженнями консалт-групи Rothman, вартість виправлення багів після випуску продукту вища в середньому в 20 разів, ніж під час повноцінного багатоетапного тестування в процесі розробки;
* *«стійкість» до змін*: жорсткий каркас з етапів розробки і умова надання тільки готового продукту визначають неможливість вносити зміни під час розробки;
* *інерційність*: на перших стадіях прогноз тимчасових і фінансових витрат може змінитися в бік збільшення, але змінити проект в сторону оптимізації витрат, зміни функціоналу або концепції до випуску готового продукту неможливо;
* *підвищений ризик*: класична система тестування має на увазі окреме тестування кожного з компонентів проекту, в тому числі, у взаємодії з іншими. При використанні Waterfall відбувається тестування готового продукту.

Частково недоліки водоспадної моделі розробки виправлені в модифікаціях Waterfall: Сашимі, Waterfall з субпроектів та водоспадна модель розробки зі зниженням ризику.

* *Сашимі або водоспадна модель з фазами, що нашаровуються* — найвідоміша серед них. У ній етапи, як і в оригінальній методиці, йдуть один за одним, але при цьому перекриваються одна іншою в часі.
* *Waterfall з субпроектів* — модель, в якій ви працюєте з трьома великими блоками: концептуалізацію, проектуванням вимог і архітектурною структурою продукту. Потім для кожного з них ви проходите стадії (субпроекти) детального проектування, кодування і тестування. В кінці проводиться інтеграція всіх компонентів на етапі тестування системи.
* *Водоспадна модель розробки зі зниженням ризику* — модифікація класичного Waterfall, в який додані спіралі зниження ризику, які поділяють проект на міні-проекти і кореспондують їх одному або декільком ключовим ризикам.

**Ітеративна розробка RUP**

Один з найвідоміших процесів, які використовують ітеративну модель розробки - Rational Unified Process (RUP). Він був створений у другій половині 1990-х років у компанії Rational Software. Основними розробниками були Філіп Крачтен (Philippe Kruchten), Грейді Буч (Grady Booch), Джеймс Рамбо (James Rumbaugh) і Айвар Якобсон (Ivar Jacobson). До речі, останні троє є також авторами нотації UML.

Термін RUP означає як методологію розробки, так і продукт компанії IBM (раніше - Rational) для управління процесами розробки. Методологія RUP описує абстрактний загальний процес, на основі якого організація або проектна команда повинна створити спеціалізований процес, орієнтований на її потреби.

Основні характеристики процесу RUP.

*Розробка вимог*

Для опису вимог RUP використовуються прецеденти використання (use cases). Повний набір прецедентів використання системи разом з логічними відносинами між ними (прецеденти можуть включати і розширювати інші прецеденти) називається *моделлю прецедентів використання*.

Кожен прецедент використання - це опис сценарію взаємодії користувача з системою, повністю виконує конкретну його завдання. Проте згідно RUP всі функціональні вимоги повинні бути представлені у вигляді прецедентів використання. Вважається, що модель прецедентів дає більш цілісне уявлення про функціональність системи в порівнянні з традиційним описом вимог (перерахуванням функцій, якими повинна володіти система).

*Ітеративна розробка*

Проект в RUP складається з послідовності ітерацій з рекомендованою тривалістю від 2 до 6 тижнів. Основною одиницею планування ітерацій є прецедент використання. Перед початком чергової ітерації визначається набір прецедентів використання, які будуть реалізовані до її завершення. Ітеративна модель дозволяє вносити необхідні зміни в вимоги, проектні рішення і реалізацію в ході проекту.

*Архітектура*

RUP - орієнтована на архітектуру методологія. Реалізація і тестування архітектури системи повинні починатися на самих ранніх стадіях проекту. RUP використовує поняття виконуваної архітектури (executable architecture) - основи додатка, що дозволяє реалізувати архітектурно значущі прецеденти використання. Основи виконуваної архітектури повинні бути реалізовані як можна раніше. Це дозволяє оцінити адекватність прийнятих архітектурних рішень і внести необхідні корективи ще на початку проекту. Таким чином, для перших кількох ітерацій необхідно вибирати прецеденти, які вимагають реалізації більшої частини архітектурних компонентів.

RUP заохочує використання візуальних засобів для аналізу і проектування. Як правило, використовується нотація і, відповідно, засоби моделювання UML (такі як Rational Rose). Модель предметної області документується у вигляді діаграми класів, модель прецедентів використання - за допомогою діаграми прецедентів, взаємодія компонентів системи між собою описується діаграмою послідовності і т д.

*Життєвий цикл проекту*

Життєвий цикл проекту RUP складається з чотирьох фаз. Послідовність цих фаз фіксована, але кількість ітерацій, необхідних для завершення кожної фази, визначається індивідуально для кожного конкретного проекту. Фази RUP не можна ототожнювати з фазами водопадом моделі - їх призначення і зміст принципово різні.

1. Початок (Inception). Фаза зазвичай складається з однієї ітерації. В ході виконання цієї фази необхідно:

* Визначити бачення і межі проекту.
* Створити економічне обґрунтування (business case).
* Ідентифікувати більшу частину прецедентів використання і докладно описати кілька ключових прецедентів.
* Знайти хоча б одне можливе архітектурне рішення.
* Оцінити бюджет, графік і ризики проекту.

Якщо після завершення першої ітерації зацікавлені особи приходять до висновку про доцільність виконання проекту, проект переходить в наступну фазу. Інвкше - його або скасовують, або проводять ще одну ітерацію фази Початок.

2. Проектування (Elaboration). У результаті виконання цієї фази на основі вимог і ризиків проекту створюється основа архітектури системи. Проектування може тривати до двох-трьох ітерацій або бути повністю пропущеним (якщо у проекті використовується архітектура існуючої системи без змін). Цілями цієї фази є:

* Детальний опис більшої частини прецедентів використання.
* Створення оттестованої (за допомогою архітектурно значущих прецедентів використання) базової архітектури.
* Зниження основних ризиків і уточнення бюджету і графіка проекту.

Основним результатом цієї фази є не безліч документів із специфікаціями, а діюча система з 20-30% реалізованих прецедентів використання.

3. Побудова (Construction). У цій фазі (триває від двох до чотирьох ітерацій) відбувається розробка остаточного продукту. Під час її виконання створюється основна частина вихідного коду системи і випускаються проміжні демонстраційні прототипи.

4. Впровадження (Transition). Цілями цієї фази є проведення бета-тестування і тренінгів користувачів, виправлення виявлених дефектів, розгортання системи на робочому майданчику, при необхідності - міграція даних. Крім того, на цій фазі виконуються завдання, необхідні для проведення маркетингу і продажів. Фаза впровадження займає від одного до трьох ітерацій. Після її завершення проводиться аналіз результатів виконання всього проекту: що можна змінити для поліпшення ефективності у майбутніх проектах?

*Робочий процес*

У термінах RUP учасники проектної команди створюють так звані артефакти (work products), виконуючи завдання (tasks) в рамках визначених ролей (roles). Артефактами є специфікації, моделі, вихідний код і т.п. Завдання розділяються по дев'яти процесним областях, званим дисциплінами (discipline). У RUP визначено шість інженерних і три допоміжні дисципліни. У них входять:

1. Бізнес-моделювання (Business Modeling) - дослідження і опис існуючих бізнес-процесів замовника, а також пошук можливих поліпшень.
2. Керування вимогами (Requirements Management) - визначення меж проекту, розробка функціонального дизайну майбутньої системи та його узгодження із замовником.
3. Аналіз і проектування (Analysis and Design) - проектування архітектури системи на основі функціональних вимог та її розвиток на протязі всього проекту.
4. Реалізація (Implementation) - розробка, юніт-тестування і інтеграція компонентів системи.
5. Тестування (Test) - пошук і відстеження дефектів у системі, перевірка коректності реалізації вимог.
6. Розгортання (Deployment) - створення дистрибутива, установка системи, навчання користувачів.
7. Управління конфігураціями і змінами (Configuration and Change Management) - управління версіями вихідного коду і документації, процес обробки запитів на зміну (change requests).
8. Управління проектом (Project Management) - створення проектної команди, планування фаз і ітерацій, управління бюджетом і ризиками.
9. Середовище (Environment) - створення інфраструктури для виконання проекту, включаючи організацію і налаштування процесу розробки.

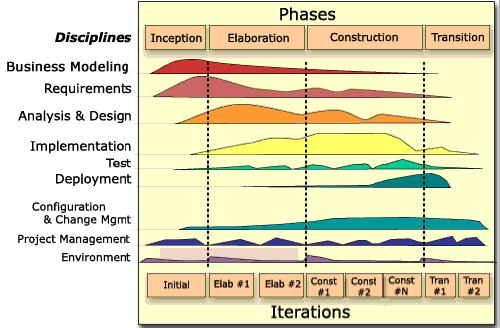
В ході життєвого циклу проекту розподіл зусиль проектної команди між дисциплінами постійно змінюється. Наприклад, як правило, на початку проекту велика частина зусиль витрачається на аналіз і дизайн, а ближче до завершення - на реалізацію і тестування системи. Однак у загальному випадку завдання зі всіх дев'яти дисциплін виконуються паралельно. Для повноцінного впровадження RUP організація повинна витратити значні кошти на навчання співробітників. При цьому спроба обійтися своїми силами швидше за все буде приречена на невдачу - необхідно шукати спеціаліста з процесів (process engineer) з відповідним досвідом або залучати консультантів.

Рис. Розподіл зусиль при виконанні проекту.

**V-Model**

Стадії розробки ПЗ тут послідовні, що "успадковане" від "водоспаду". Модель хороша для тих систем, де потрібно безперебійне функціонування. Наприклад: створення прикладного ПЗ для клінік, використовуваного для безперервного спостереження за пацієнтами; ПЗ, яке керує механізмами подушок безпеки в транспортних засобах.

Проект виконується при цьому за чітким пунктами творчого завдання. Однак значна роль приділяється і своєчасного тестування: функціональному, інтеграційному, навантаженню, зручності інтерфейсу.

Коли необхідно використовувати дану методологію для розробок. У випадках, де потрібна проведення ретельного тестування продукту. Для невеликих і середніх проектів з чітко визначеними вимогами. В умовах, коли доступні інженери, тестувальники конкретної кваліфікації.

**Incremental Model - Інкрементна модель**

У цій технології розробки ПЗ комплекс вимог до системи розділяється на складові, які є описом поетапної зборки. Кілька циклів розробки проекту складаються в комплекс, іменований "мульти-водоспад".

Цикл, в свою чергу, поділяється на окремі легко створювані модулі. Кожен з них проходить через етапи визначення вимог, проектування, впровадження, тестування, кодування.

Особливість тут у тому, що на першому великому етапі випускається базова модель розробки. А потім до неї додаються инкременты - нові функції. Такий процес триває до тих пір, поки не створюється повний комплекс. В додаткових моделях окремі запити постають ясними, можуть бути просто формалізовані і реалізовані.

Використання Incremental Model буде обґрунтованим у таких випадках. Чітко визначені і зрозумілі вимоги до кінцевого продукту. Допускається доопрацювання деяких деталей з плином часу. Є декілька ризикових цілей. Необхідний ранній висновок на ринок.

**RAD Model** (від англ. *rapid application development*) - **Швидка́ розро́бка застосу́нків**

RAD Model виступає однією з різновидів інкрементній моделі. Компоненти або функції програми тут паралельно розробляються кількома командами професіоналів. У результаті виходить кілька міні-проектів. Час на створення кожного з них жорстко обмежений. Всі модулі складаються в загальний робочий прототип. Такий підхід допомагає швидко представити замовнику для огляду робочий продукт, який потім можна внести ряд змін.

Процес розробки ПЗ включає в себе кілька етапів:

1. Бізнес-моделювання. Це визначення інформаційних потоків між спектром підрозділів.

2. Моделювання відомостей. Дані, зібрані на першому етапі, використовуються для визначення сутностей, необхідних для циркуляції інформації.

4. Моделювання процесу. Під час цієї фази інформаційні потоки пов'язують певні об'єкти для досягнення мети розробки.

5. Складання програми. Використовуються засоби автоскладання для перетворення моделей проектування в код.

6. Тестування. Перевірка нових компонентів і інтерфейсів.

Використовувати такий метод розробки ПЗ слід тільки в тому випадку, коли в команді є висококваліфіковані та "вузькі" фахівці. Бюджет проекту, безумовно, великий: потрібно оплатити роботу професіоналів, вартість готового інструментарію автоматизованої збірки.

Модель вибирають при впевненому знанні цільового бізнесу в тих випадках, коли потрібно представити готовий продукт в короткі терміни - за 2-3 місяці.

**Iterative Model - Ітеративна модель**.

Особливістю проекту є те, що для початку його реалізації не потрібна повна специфікація вимог. Створення починається з конструювання бази, яка повинна стати основою для визначення подальших вимог.

Версія в даному випадку може бути і зовсім неідеальною. Головна вимога - щоб вона працювала. Розробник розуміє і бачить кінцеву мету роботи. Він повинен прагнути до того, щоб кожен крок його діяльності був результативний, а кожна створена версія працездатна.

Застосування ітеративної моделі виправдана в таких випадках. Вимоги до кінцевої версії розробки зрозумілі та чітко визначені. Проект дуже масштабний. Основне завдання заздалегідь визначена. Але її деталі допустимо удосконалювати, змінювати у процесі роботи.

**Spiral Model Спіральна модель**.

Акцент робиться на оцінку ризиків. Найбільше дана методологія може бути застосована для вирішення критичних бізнес-завдань, коли неуспішність проекту може серйозно зашкодити діяльності компанії. Спіральну модель широко застосовують при випуску нових лінійок ПЗ, при необхідності проведення наукових досліджень проекту, практичної апробації. Кожен з "витків спіралі" проходить чотири фази: Планування. Аналізування ризиків. Конструювання. Оцінка підсумків. Якщо вона позитивна, то розробник переходить до нового "витка" проекту.

Спіральну модель не варто використовувати для маленьких бюджетних проектів. Навпаки, вона більше підходить для масштабних і дорогих, наприклад, для розробки системи банківського документообігу. Тут велика увага приділяється не стільки самому програмування, скільки аналізу кожного вже виробленого "витка".

**Адаптивні методології**

Адаптивні методології націлені на подолання очікуваного недосконалості, неповноти вимог, на постійну зміну останніх. Відповідно, зі зміною вимог може замінюватися і команда розробників проекту. Точний план з адаптивної методології розробляється лише на найближчий час. Плани, пов'язані з більш віддаленим від реальності подій, існують у формі декларацій про мету роботи, її результати та очікуваних витратах. До адаптивним відносяться всі методології, які задовольняють вимогам, сформульованим у Маніфесті адаптивної розробки. Серед адаптивних методологій: (Scrum, Crystal, Extreme Programming, Adaptive Software Development, DSDM, Feature Driven Development, Lean software development).

Agile Manifesto був розроблений і прийнятий 1-13 лютого 2001 року у лижному комплексі в горах Юти. Містить в себе 4 головні ідеї і 12 принципів командної роботи без єдиного практичного поради.

Основні ідеї:

1. Взаємодія і люди найголовніше інструментів і процесів.
2. Працюючий продукт вище вичерпної документації.
3. Співпраця з клієнтом найголовніше узгодження окремих умов контрактів.
4. Готовність команди до змін важливіше проходження початкових планів.

Принципи діяльності розробників:

1. Найвищим пріоритетом є задоволення потреб замовника, шляхом завчасного та регулярного постачання програмного забезпечення.
2. Схвальне ставлення до змін, навіть на заключних стадіях розробки. Agile-процеси надають можливість використовувати зміни задля забезпечення конкурентоспроможності замовника.
3. Працюючий продукт слід випускати якомога частіше, з періодичністю від пари тижнів до пари місяців.
4. Впродовж усього проекту розробники і представники бізнесу повинні працювати разом щодня.
5. Над проектом повинні працювати вмотивовані професіонали. Щоб робота була виконана, створіть їм умови, надайте підтримку і повністю на них покладіться.
6. Особиста комунікація – найефективніший та найпрактичніший метод як донести інформацію до команди, так і поширити її всередині.
7. Працюючий продукт – головний показник прогресу.
8. Інвестори, розробники і користувачі повинні мати можливість підтримувати постійний ритм як завгодно довго. Agile допомагає налагодити такий сталий процес розробки.
9. Постійна увага до технічної досконалості і якості проектування підвищує гнучкість проекту.
10. Простота – мистецтво мінімізації зайвої роботи – вкрай необхідна.
11. Найкращі вимоги, архітектурні та технічні рішення виникають у командах, що здатні самоорганізовуватись.
12. Команда регулярно намагається знайти способи підвищення ефективності та відповідно корегує свою роботу.

Гнучкі методології розробки програмного забезпечення (від англ. Agile software development) або agile-методи - це комплекс підходів до розробки ПЗ, що орієнтований на використання ітеративных розробок, динамічне формування вимог до проекту, забезпечення реалізації в результаті безперервної взаємодії всередині робочих самоорганізованих груп, складених з фахівців різного профілю.

Гнучкі методології розробки програмного комп'ютерного забезпечення спрямовані на мінімізацію ризиків шляхом приведення спільного проекту до комплексу коротких циклів - ітерацій, кожен з яких максимально триває 2-3 тижні. Ітерація тут - мініатюрний програмний проект, який включає в себе всі завдання по забезпеченню функціонального міні-приросту в частині планування, аналізування вимог, проектування, програмування, тестування розробки, документування. До кінця кожної ітерації готовий «гнучкий» програмний продукт, і команда обов'язково виконує переоцінку пріоритетів розробки. Під час кожної ітерації робиться наголос на безпосередню комунікацію фахівців "лицем до лиця" (тестувальників, технічних письменників, дизайнерів інтерфейсу та ін.). Більшість команд розташовується в одному офісі. Обов'язково присутність "замовника", який пред'являє вимоги до розробки. Основним показником тут виступає кінцевий продукт. За рахунок безпосереднього спілкування фахівців тут порівняно маленький обсяг супутньої письмовій документації.

### 9 основних методологій Agile розробки ПЗ

**Agile Modeling**

Набір концепцій, принципів і методів (практик), який дозволяє швидко і легко виконувати проектування та документацію для проектів з розробки програмного забезпечення. Не включає в себе докладні інструкції з проектування, містить описи, як побудувати діаграми UML. Основна мета – ефективне моделювання та документація, але не включає в себе програмування і тестування, управління проектом, розгортання та обслуговування системи.

**DSDM**

Грунтується на концепції швидкої розробки додатків (Rapid Application Development, RAD). Це ітеративний і інкрементний підхід, який підкреслює постійну участь користувача/споживача в проекті.

**Extreme programming (XP)**

Ідея полягає в тому, щоб використовувати корисні традиційні методи розробки по-новому. Наприклад, код що написаний одним розробником, перевіряється іншим. Також дуже часто використовують парне програмування, де один кодер пише код, а його партнер цей код одразу перевіряє.

З усіх гнучких методологій методологія екстремального програмування (XP) — найбільше відома. *XP стоїть на чотирьох китах: Комунікація, Зворотній зв'язок, Простота і Сміливість.* Із них слідують дванадцять практик, яким повинні слідувати проекти, що використовують ХР. Багато з цих практик являють собою старі перевірені техніки, які, тим не менш, багато встигли забути (включаючи більшість передбачуваних процесів). ХР не тільки воскрешає до життя такі техніки, але і з'єднує їх таким чином, що всі вони підтримують і підсилюють один одного.

На цьому фундаменті ХР будує еволюційний процес проектування, заснований на реорганізації коду системи протягом кожної наступної ітерації. При цьому проектується тільки та функціональність, яка відноситься до поточної ітерації, а будь-які майбутні потреби не враховуються. Одержаний в результаті процес вимагає від розробників дисципліни, і в той же час поєднує її з високою адаптивністю. Таке дивне поєднання дозволяє припустити, що ХР є найбільш розвиненою адаптивної методології.

**Feature driven development (FDD)**

Функціонально-орієнтована розробка. Поняття FDD дуже близькі до RUP, єдина різниця: “Кожна функція повинна реалізовуватися не більше двох тижнів.” Якщо завдання досить невелике, його можна розглядати як окрему фунцію. Якщо завдання є великим, то він повинен бути розділений на декілька відносно самостійні функції.

FDD забезпечує масштабованість і повторюваність. Але при цьому заохочується творчий підхід, застосування в роботі інновацій. Основні принципи методології наступні:

* Розробка кожного великого проекту - це системна діяльність.
* Всі процеси повинні бути простими і добре проробленими.
* Логічність і цінність кожного з процесів повинна бути зрозуміла будь-кому з членів команди.
* Перевагу - коротким циклам розробки ПЗ. Це дозволяє знизити кількість помилок, разом з тим збільшуючи функціональність.

Цінність методології і в тому, що вона чітко регламентує тривалість процесів. При цьому на організаційні питання в кожному циклі не повинно витрачатися більше 25 % часу. Решта 75 % - суто на розробку, складання, тестування функціоналу.

**OpenUP**

Цей метод розробки є ітераційно-інкрементний. Позиціонується як одна із версій RUP. OpenUP ділить життя проекту на чотири фази: початковий етап, фаза специфікації, розробка та передача готового фунціоналу. Життєвий цикл проекту надає зацікавленим особам та членам колективних точку відліку і прийняття рішень протягом всього проекту. В цьому випадку можливо ефективно контролювати процес розробки.

**Scrum**

Встановлює правила для процесу управління розробки програмного забезпечення і дозволяє використовувати існуючу практику кодування, регулювати вимоги або приймати тактичні зміни. Використовуючи цю методику можливо виявити і усунути відхилення від бажаного результату на більш ранніх стадіях розробки програмного забезпечення.

Scrum надає емпіричний підхід до розробки ПО. Цей процес швидкий, адаптивний, вміє самонастриваться і відмінний від послідовного (водопадного) підходу. Scrum заснований на повторюваних циклах, це робить його більш гнучким і передбачуваним.

В основі лежать короткі щоденні зустрічі - Scrum і циклічні 30-денні зустрічі, звані Sprint. Під час щоденного Scrum'a кожному члену команди задаються тільки 3 питання:

Що було зроблено з останнього Scrum'a?

Що ви будете робити між цією і наступний зустріччю Scrum?

Що заважає вам виконувати роботу?

Головні принципи Scrum

* Індивідуалізм і взаємодія важливіше суворих процесів і методів.
* Робота продукта важливіше ЗА складну документацію.
* Взаємодія з замовниками важливіше контрактних домовленостей.
* Готовність змінити важливіше дотримання плану.

Scrum - це мистецтво можливого.

Девізом Scrum'a можна вважати дотримання здоровому глузду.

Побудова команди

Команда - головна рушійна сила проекту, це джерело продуктивності та креативності. Команда, яка буде брати участь у щоденних Scrum'ах повинна складатися з 7 плюс-мінус 2 людини. Більше число не рекомендується, краще розділити на кілька команд. Scrum Master (або, більш зрозуміло - Project manager) веде цю зустріч. Члени команди називаються "свинями" і тільки вони мають право говорити. На зустрічах можуть бути присутніми мовчазні відвідувачі - "курки", які такого права позбавлені. Scrum зустрічі повинні бути короткими, запізнення не вітаються. Кен Швабер (Ken Schwaber, автор книги "Agile Project Management with Scrum") на своїх тренінгах пропонував ввести практику грощове покарання, коли кожен спізнився платить долар або євро, зібрані гроші згодом він віддавав бездомним на вулиці.

В команді не повинні бути тільки розробники. Команда повинна включати в себе також і тестерів, дизайнерів, а також інших зацікавлених осіб. Бажано, якщо робочі місця всієї команди знаходяться в одній кімнаті, або, як мінімум у сусідніх. Це повинно сприяти максимальному комунікації між членами команди.

Scrum Master і управління проектом.SCRUM-структура

Оскільки в основі Scrum'a лежать щоденні зустрічі і 30-денні циклічні зустрічі, звані Sprint, всі виникаючі організаційні і технічні проблеми всім добре видно.

В кінці кожного Sprint'a команда повинна демонструвати замовнику готову програму. Вимоги щодо функціональності повинні бути визначені на початку цього Sprint'a місяць тому і не повинні були змінюватися протягом цього часу. Ці вимоги відображаються в бэклоге - окремій таблиці зі вказівкою того, що потрібно зробити і скільки це займе часу.

Успіх Scrum'a залежить багато в чому до діяльності Scrum Master'a. У числі головних завдань і особливостей діяльності:

* Видаляти бар'єри між замовником і розробниками, так щоб замовник безпосередньо керував розробкою.
* Довіряти своїй команді, робити все можливе, щоб розкрити креативність кожного її члена.
* Збільшувати продуктивність команди будь-якими можливими способами.
* Мертвий Scrum Master - даремний Scrum Master.
* Поганий досвід буде помітний відразу, хороший - непомітний.
* Вирішення конфліктів.

Висновки

1. Це промисловий процес. В даний час в ньому немає місця одиноким геніям та замкнутих команд. Це анахронізм.

2. Місце творчості займає чіткий рух до вказаної мети. Творчість ж можна виявити, обравши найбільш зручний спосіб досягнення цієї мети.

3. Удосконалення процесів у компанії повинно відбуватися щодня. Потроху, але постійно. Одна з основних причин постійної роботи над поліпшенням те, що без постійної підтримки процеси мають властивість деградувати, причому досить швидко.

4. Поліпшення - справа кожного співробітника. Грамотно поставлені процеси і дотримання прийнятої в компанії методології полегшують життя, а не ускладнюють її. І якщо у змін немає підтримки "знизу", то провал змін, ініційованих топ-менеджментом - неминучий.

**Lean software development**

Ця методологія розробки програмного забезпечення метод бережливого виробництва (Lean Manufacturing) або LD *(Lean software development)*, ощадлива розробка ПЗ. Основні принципи цієї методології є: максимальна швидка доставка продукції замовнику з дуже короткими ітераціями, мотивація команди і зосередитися на навчанні з короткими циклами розвитку, раннє тестування і зворотній зв’язок з клієнтами.

Головне достоїнство LD: збереження високого морального і функціонального стану фахівців, а саме:

* Заохочення кожного з працівників за особливо успішну діяльність.
* Поточні завдання проекту змінюються лише в разі крайньої необхідності або за бажанням замовника.
* Суворе виконання плану.
* Сверхроботи вважаються ознакою втрати як часу, так і ресурсів. провадження в роботу загальній концепції діяльності: "Широко мислити, швидко помилятися, мало працювати, стрімко навчатися".

**Kanban software development**

Канбан реалізує принцип «точно в строк» і урівноважує робоче навантаження між усіма членами команди. За допомогою цього методу весь процес розробки зрозумілий для всіх членів команди. Канбан є візуальною моделлю розвитку, яка показує те, що потрібно виробляти, коли і скільки.

### Шість ключових практик методу Канбан

1. Візуалізуйте.

Візуалізація процесів роботи допомагає в правильному розумінні змін, що плануються, і допомагає впроваджувати їх згідно з планом. Типовим способом візуалізувати процес роботи є використання дошки з колонками і картками. Колонки на дошці позначають різні кроки процесу роботи.

1. Обмежуйте задачі в процесі виконання

Обмеження задач в процесі виконання має на увазі те, що використовується система «витягування» на частинах, або всьому процесі роботи. Система «витягування» працює як один з головних стимулів до постійних покращень в системі. Система «витягування» може бути реалізована як система канбан чи якийсь інший варіант. Критичним елементом є те, що робота, котра перебуває в стані виконання на кожному кроці робочого процесу, є обмеженою, і що нова робота «витягується» в кожен крок, коли з'являється місце в колонці кроку.

1. Керуйте потоком.

Кожен перехід між станами в потоці моніториться, вимірюється і звітується. Активне управління потоком дозволяє оцінити позитивні та негативні ефекти змін у системі.

1. Зробіть політики явними .

Поки механізм чи процес не стане явним, часто важко чи неможливо здійснювати обговорення щодо його вдосконалення. Без явного розуміння, як все працює, будь-які обговорення проблем стають емоційними та суб'єктивними. З явним розумінням можливо перейти до більш раціональних, емпіричних та об'єктивних обговорень проблем.

1. Створіть цикли зворотнього зв'язку

Організації що не створили другий рівень зворотнього зв'язку — перегляд операцій, — зазвичай не бачать вдосконалення процесу поза локалізованим рівнем команди.

1. Вдосконалюйте співпрацюючи, розвивайтесь експериментально (використовуючи моделі та науковий метод)

Метод Канбан пропагує малі поступові, постійні та еволюційні зміни які приживаються. Коли команди мають спільне розуміння теорій про роботу, процес, ризики, вони більш ймовірно будуть здатними виробити спільне розуміння проблем та запропонувати вдосконалення які будуть результатом консенсусу. Метод Канбан радить використовувати науковий підхід до втілення змін.

**Scrumban**

Scrumban є структурою управління і гібрид Scrum і Kanban. Розробники працюють з історіями користувачів і намагаються зберігати ітерації якомога коротшими. Тут не існує конкретних ролей, як наприклад в Scum. Кожен член команди зберігає свою існуючу роль в проекті.

Тут також деякі інші методології розробки програмного забезпечення по Agile: **Agile Unified Process (AUP), Agile Data Method, Essential Unified Process (EssUP), Getting Real**

### Open Source

У більшості проектів з відкритим вихідним кодом є один або кілька координаторів. Координатор є лідером проекту, єдиною людиною, яка може вносити зміни безпосередньо в репозиторій вихідного коду. Тим не менш, інші розробники теж можуть вносити код зміни, з тієї лише різницею, що їм доведеться спочатку відіслати їх координатору, який прогляне виправлений код і вже потім вносить зміни в репозиторій. Зазвичай такі зміни мають вигляд патч-файлів, що спрощує подібну процедуру. Таким чином, лідер проекту координує патчі і стежить тим, щоб вони відповідали загальним планом розроблюваного ПЗ.

Ще однією особливістю розробки проектів з відкритим вихідним кодом є те, що налагодження програми може вестися паралельно. Таким чином, в ній можна задіяти велику кількість людей. Знайшовши в програмі помилку, вони можуть відіслати патч лідеру проекту. Таким чином, не-координатори виконують дуже важливу функцію, тому що більша частина часу витрачається саме на пошук помилок. Крім того, ця робота підходить тим, у кого немає хороших навичок проектування.

***Контрольні запитання*.**

1. За якими принципами можна класифікувати методології розробки ПЗ?

2. Чому каскадна модель відноситься до прогнозованих методологій?

3. Охарактеризуйте методологію RUP?

4. Які основні ідеї і принципи "гнучких" методологій?

5. Де краще застосовувати "гнучкі", а де прогнозовані методології?

6. В чому сутність методології RAD?

***Рекомендована література***

1. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
2. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
3. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
4. Проектування інформаційних систем: Посібник // За редакцією Пономаренка В.С. – К.: Видавничий центр "Академія". 2002. ­ 488 с. URL: <http://www.dut.edu.ua/uploads/l_874_10304054.pdf>.
5. Якобсон А. Унифицированный процесс разработки программного обеспечения / А.Якобсон, Г.Буч, Д.Рамбо. –СПб.: Питер, 2002. – 496 с.
6. Блог PM Рішення: Що таке канбан і чим він корисний? URL: https://worksection.com/ua/blog/kanban.html

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.