ТЕМА 2. МОДЕЛІ ПРОГРАМНИХ ПРОЕКТІВ З ІТЕРАТИВНИМ ЖИТТЄВИМ ЦИКЛОМ

Найбільш поширені об‘єктно-орієнтовані моделі, орієнтовані на ітеративний процес розроблення – Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Personal Software Process / Team Software Process (PSP/TSP) і Agile (eXtreme Programming (XP), Crystal, Feature Driven Development (FDD), Scrum).

1. Методологія Rational Unified Process

Уніфікований процес Rational Unified Process (RUP) був розроблений Філіпом Крачтеном (Philippe Kruchten), Іваром Якобсоном (Ivar Jacobson) та іншими співробітниками компанії «Rational Software» як доповнення до мови моделювання Unified Modeling Language (UML). Особливістю RUP є те, що в результаті роботи над проектом створюються та вдосконалюються моделі ПЗ. Замість створення величезної кількості паперових документів, RUP спирається на розробку і розвиток семантично збагачених моделей, які всебічно представляють систему, що розробляється.

RUP, на відміну від більшості інших методологій, дозволяє в широкому діапазоні обирати ступінь формалізації і ітеративний процес розроблення в залежності від особливостей проектів і організації. RUP найбільш часто використовують компанії з великою кількістю розробників (понад 40-50 чоловік).

ЖЦ проекту RUP складається фаз і дисциплін. Фази RUP: початок (Inception), проектування (Elaboration), розробка (Construction) та впровадження (Transition). Послідовність цих фаз фіксована, але число ітерацій, необхідних для завершення кожної фази, визначається індивідуально для кожного конкретного проекту. Кожна фаза складається з дисциплін: «Бізнес-моделювання» (Business modeling); «Керування вимогами» (Requirements); «Аналіз і Проектування» (Analysis and Design); «Реалізація» (Implementation); «Тестування» (Test); «Розгортання» (Deployment); «Керування проектом» (Project management); «Керування змінами» (Change management); «Середовище» (Environment).

У RUP всі дисципліни виконуються практично у всіх фазах ЖЦ ПЗ. Однак, в залежності від фази, змінюються поточні цілі проекту і співвідношення між обсягами робіт, що відповідають різним матеріалам.

Розглянемо рольову структуру проекту, що запропонована Центром об‘єктно-орієнтованої технології компанії IBM:

– замовник (Customer) реально існуючий (в організації, якій підпорядкована команда або поза нею) ініціатор розроблення або хто- небудь інший, уповноважений приймати результати (як поточні, так і остаточні) розроблення;

– планувальник ресурсів (Planner) висуває та координує вимоги до проектів в організації, що здійснює цю розробку, а також розвиває та спрямовує план виконання проекту з точки зору організації;

– менеджер проекту (Project Manager) відповідає за розвиток проекту в цілому, гарантує, що розподіл завдань і ресурсів дозволяє виконати проект, що роботи та подання результатів йдуть за графіком, що результати відповідають вимогам. У рамках цих функцій менеджер проекту взаємодіє з замовником і планувальником ресурсів;

– керівник команди (Team Leader) виконує технічне керівництво командою в процесі виконання проекту. Для великих проектів можливе залучення декількох керівників підкоманд, які відповідають за вирішення своїх завдань.

– архітектор (Architect) відповідає за проектування архітектури системи, погоджує розвиток робіт, пов‘язаних з проектом.

– проектувальник підсистеми (Designer) відповідає за проектування підсистеми або категорії класів, визначає реалізацію та інтерфейси з іншими підсистемами.

– експерт з предметної області (Domain Expert) відповідає за вивчення сфери застосування, підтримує спрямованість проекту на вирішення завдань цієї галузі;

– розробник (Developer) реалізує спроектовані компоненти, має та створює специфічні класи та методи, здійснює кодування та автономне тестування, будує продукт. Це широке поняття, що може підрозділятися на спеціальні ролі (наприклад, розробник класів). Залежно від складності проекту, команда може включати різну кількість розробників;

– розробник інформаційної підтримки (Information Developer) створює документацію, що супроводжує продукт, коли випускається версія. До неї включаються інсталяційні, інформаційні та навчальні матеріали. Матеріали допомоги надаються на паперових і машинних носіях. Для складних проектів можливий розподіл цих завдань між кількома розробниками інформаційної підтримки;

– спеціаліст з інтерфейсу користувача (Human Factors Engineer) відповідає за зручність застосування системи. Працює з замовником, щоб упевнитися, що призначений для користувача інтерфейс задовольняє вимогам;

– тестувальник (Tester) перевіряє функціональність, якість та ефективність продукту; будує та виконує тести для кожної фази розвитку проекту;

– бібліотекар (Librarian) відповідає за створення та ведення спільної бібліотеки проекту, що містить всі проектні робочі продукти, а також за відповідність робочих продуктів стандартам.

Методологія RUP використовує програмні рішення Rational Application Developer, Rational Rose, Rational Asset Manager, Rational Software Architect, Rational Software Modeler, Rational Suite, Rational Requisitepro, Rational Clearquest, Rational Clearcase та інші.

Переваги RUP:

– дуже широкий діапазон рішень в частині формалізації процесу розроблення;

– можна використовувати як основу для водоспадного стилю розроблення, так і в якості гнучкого процесу;

– зниження основних ризиків замовника і розробника;

– економія ресурсів за рахунок автоматизації регресійного тестування;

– поліпшення якості ПЗ за рахунок багаторазових перевірок змін;

– поліпшення якості тестування за рахунок використання сучасних технологій.

Недоліком є недостатній рівень формалізму, який може призводити до неузгодженості рішень, прийнятих учасниками проекту, до непродуктивних витрат ресурсів на переробку коду і на повторне рішення типових проблем.

2. Модель Microsoft Solutions Framework

У 1994 році Microsoft випустила MSF — набір концепцій і рекомендованих моделей, які дозволяють розробляти і впроваджувати розподілені інформаційні системи масштабу підприємства на основі технологій та інструментальних засобів фірми Microsoft. MSF базується на практичних результатах організації розподілених обчислень і застосування клієнт-серверних технологій, отриманих як у фірмі Microsoft, так і її партнерами і замовниками.

Процес розроблення відповідно до MSF є ітеративним і включає в себе такі основні фази: розробка концепції (єдиного бачення), планування, розробка, стабілізація (тестування), впровадження. Кожна фаза циклу закінчується головною віхою (контрольної точкою). Відповідно головні віхи будуть мати назви: концепція продукту затверджена, плани продукту затверджені, розробка завершена, готовність рішення затверджена, впровадження завершено. Віха є точкою синхронізації досягнутих результатів і очікувань замовника, а також аналізу проектного середовища. У рішенні про закриття чергової фази повинні приймати участь відповідальні представники всіх рольових кластерів (розробка, тестування, впровадження, керування проектом тощо).

MSF пропонує досить нестандартні підходи до організаційної структури, розподілу відповідальності і принципів взаємодії всередині команди. MSF пропонує розбиття великих команд (більше 10 осіб) на малі багатопрофільні групи напрямів (feature teams). Ці малі колективи працюють паралельно, регулярно синхронізуючи свої зусилля.

Microsoft пропонує інструменти для реалізації MSF: MSF for Agile Software Development, MSF for CMMI Process Improvement, Visual Studio Team System (Team Suite; Team Edition for Database Professionals; Team Suite for Software Architects; Team Suite for Software Developers; Team Suite for Software Testers; Team Foundation Server; Team Test Load Agent).

Перевагами MSF є:

– систематизація та структуризація інформації у формі бази знань;

– нестандартні підходи до організаційної структури, розподілу відповідальності і принципів взаємодії всередині команди;

– легка масштабованість MSF — мінімальний розмір проектної групи в MSF-проекті 3 людини, але застосовувати дану методологію можна для колективів і в десятки, сотні, тисячі чоловік;

– не нав‘язливість MSF щодо використання будь-яких конкретних інструментів і програмних засобів.

Недоліки MSF: не описує детально найважливіші ролі замовника і користувача; не розглядаються методи керування групою проектів.

3. Модель Personal Software Process/Team Software Process

В кінці 90-х років SEI розробив модель PSP / TSP, засновану на моделі зрілості CMM. PSP визначає вимоги до компетенцій розробника. Відповідно до цієї моделі кожен програміст повинен вміти: враховувати час, витрачений на роботу над проектом; враховувати знайдені дефекти; класифікувати типи дефектів; оцінювати розмір задачі; здійснювати систематичний підхід до опису результатів тестування; планувати програмні завдання; розподіляти їх за часом і складати графік роботи; виконувати індивідуальну перевірку проекту та архітектури; здійснювати індивідуальну перевірку коду; виконувати регресійне тестування.

PSP включає 4 процеси, які послідовно додаються розробником: PSP0 – базовий власний процес (The Baseline Personal Process); PSP1 – планування (Personal Planning Process); PSP2 – керування якістю (Personal Quality Management); PSP3 – циклічний процес (Cyclic Personal Process) .

TSP робить ставку на самокеровані команди чисельністю 3-20 розробників. Команди повинні встановити власні цілі, скласти свій процес і плани, відстежувати роботу, підтримувати мотивацію і максимальну продуктивність.

Методологія є ефективною, може забезпечити істотні переваги і стати моделлю для будь-якої організації, зацікавленої у вдосконаленні своїх процесів розроблення.

PSP / TSP — це чиста методологія щодо забезпечення якості проектів, розрахована на використання UML при проектуванні і будь- які CASE-засоби, зокрема вони орієнтовані на використання продуктів компанії Microsoft.

Переваги: найкраще планування часу та бюджету, керування якістю продукту, зменшення часу розроблення, акуратний збір метрик, оцінка часу роботи через оцінку

обсягу артефактів, використання історичних даних у плануванні, раннє виявлення дефектів.

Недоліки: складність в обліку ризиків, обліку недооцінки складності.

4. Гнучка методологія розроблення програмного забезпечення (Agile)

У лютому 2001 року 17 розробників різних методологій зібралися для того, щоб спробувати виявити що-небудь спільне у своїх підходах. Результатом став Маніфест гнучкого розроблення. Тоді ж з'явився термін Agile («гнучкий, спритний»), що об'єднує всі методології.

Agile – гнучка методологія розроблення, це концептуальний каркас, в рамках якого виконується розробка програмного забезпечення. Основна ідея всіх гнучких моделей полягає в тому, що процес, який застосовується у розробці ПО, повинен бути адаптивним. Вони декларують своєю вищою цінністю орієнтованість на людей та їх взаємодію, а не на процеси і засоби.

CASE-засоби, що можуть допомогти впровадити і поліпшити процес створення ПЗ на базі Agile: XPArchitect; Borland ALM (зокрема Silk).

Розглянемо більш детально кілька гнучких методологій: eXtreme Programming, Crystal, Feature Driven Development та Scrum.

4.1 Екстремальне програмування eXtreme Programming

Екстремальне програмування (eXtreme Programming, або XP – методологія, розроблена Кентом Беком (Kent Beck), Уордом Каннінгемом (Ward Cunningham) і Роном Джеффріс (Ron Jeffries), є сьогодні найбільш відомою з гнучких методологій. XP проповідує комунікабельність, простоту, зворотний зв'язок. Вона описується як набір практик: гра в планування, короткі релізи, метафори, простий дизайн, переробки коду (refactoring), розробка «тестами вперед», парне програмування, колективне володіння кодом, 40-годинний робочий тиждень, постійна присутність замовника і стандарти коду. При використанні XP ретельне попереднє проектування ПО замінюється, з одного боку, постійною присутністю в команді замовника, а з іншого – регулярними переробками коду (так званий рефакторинг). Основою проектної документації вважається ретельно прокоментований код. Дуже велика увага в методології приділяється тестуванню.

Життєвий цикл проекту в XP складається з послідовності релізів. Кожен реліз – це повноцінна версія продукту, яку може використовувати замовник і яка містить додаткову функціональність у порівнянні з попереднім релізом. Реліз з'являється в результаті однієї або декількох ітерацій, що тривають від одного до чотирьох тижнів.

У XP не рекомендується витрачати багато часу на планування; сам процес планування називається грою (planning game). Докладний план складається тільки на чергову ітерацію і найближчі один-два релізи.

Планування релізу складається з наступних кроків:

– замовник формулює свої вимоги у вигляді історій, які оцінюються за трудомісткістю розробниками. Оцінки трудомісткості робляться в так званих ідеальних днях – часі, який якийсь уявний розробник витратить на реалізацію історії при повній відсутності відволікаючих чинників, паралельних завдань, перерв тощо;

– історії сортуються за пріоритетом, ризикам, складністю реалізації;

– визначається фактична продуктивність команди (яке число реальних днів відповідає ідеальному дню);

– на підставі фактичної продуктивності визначається, які історії увійдуть в черговий реліз і за який час він буде завершений.

Ітерація планується аналогічним чином:

– попередньо визначається набір історій для ітерації;

– історії розбиваються на завдання (tasks), які розподіляються між розробниками. На відміну від історій, які описують поведінку системи з точки зору користувача, завдання складаються на технічному рівні, наприклад «модифіковані схему бази даних» або «провести рефакторинг»;

– розробники оцінюють свої завдання. На цьому етапі уточнені оцінки повинні бути більш реалістичними. Оскільки завдання тепер оцінюють їхні безпосередні виконавці, в оцінках враховується індивідуальна продуктивність, що залежить від досвіду, знайомства з використовуваними технологіями тощо;

– на основі уточнених оцінок завдань підраховується загальне завантаження команди. Залежно від завантаження деякі історії можуть бути перенесені на наступну ітерацію або, навпаки, додані в поточну.

Періодично в ході проекту вимірюється фактична продуктивність команди. Якщо вона починає сильно відрізнятися від значення, яке було використано при плануванні, графік виходу релізів повинен бути переглянутий.

Недоліки:

– XP можуть ефективно використовувати тільки в команді досвідчених розробників;

– команду не можна розбивати на кілька частин, можливий розмір команди обмежений числом в 10-15 чоловік, тому що велику роль в XP грає пряме спілкування;

– не завжди можна забезпечити постійну присутність представника замовника у проектній команді;

– не бажано використовувати XP в проекті з фіксованою ціною.

Незважаючи на всі перераховані обмеження, XP може показати виняткову ефективність завдяки вкрай низьким накладним витратам.

4.2 Сімейство методологій Crystal

Crystal – сімейство методологій, що визначають необхідний ступінь формалізації процесу розроблення в залежності від кількості учасників і критичності завдань. Алістер Коуберн (Alistair Cockburn) на початку 90-х на замовлення від компанії IBM написав роботу на тему методології розроблення ПЗ. При цьому його підхід суттєво відрізнявся від підходу більшості інших методологів. Його теорії засновані не тільки на особистому досвіді, а й на постійних дослідженнях інших проектів і процесів.

Ступінь важливості наростає по вертикальній осі. Величина команди наростає по горизонтальній осі. У результаті виходить сімей- ство методологій. Чим нижче критичність і чим менше команда, тим більш «легку» методологію потрібно використовувати. Найлегшою з усього сімейства є методологія Crystal Clear. Головні принципи даної методології – вся команда розробників (до 6 осіб) знаходиться в одному приміщенні;

– часті поставки продукту дозволяють виробити «ритм» проекту;

– обмін інформацією з реальними користувачами дозволяє швидко отримувати зворотній зв'язок і ліквідувати недоліки на ранніх стадіях;

– засоби контролю версій коду забезпечують колективне володіння кодом.

Сімейство методологій Crystal побудовано на ітеративної розробці. Тривалість ітерації може змінюватися в межах від 1 до 4 місяців. Чим менше ітерація, тим краще.

Важливою особливістю Crystal є безперервне налагодження методології. Це дозволяє поступово адаптувати її для конкретної команди і конкретного проекту. В жодній іншій методології налаштуванню не приділяється такої уваги.

Перевагою Crystal Clear є те, що вона максимально проста у використанні, вимагає мінімальних зусиль для впровадження, орієнтована на людські звички, описує природний порядок розроблення ПЗ.

Недоліки:

– поступається XP за продуктивністю;
– дуже складно заздалегідь передбачити, які проміжні продукти необхідні.

4.3 Методологія Feature Driven Development

Методологія Features Driven Development FDD була розроблена Джеффом Де Люка (Jeff De Luca) і Пітером Коадом (Peter Coad) в 1997 році. Це функціонально-орієнтована розробка, яка оперує поняттям функції або властивості (feature) системи, досить близьким до поняття сценарію використання, вживаному в RUP. відмінність – це додаткове обмеження: «кожна функція повинна допускати реалізацію не більше ніж за два тижні».

FDD включає п'ять процесів, причому останні два повторюються для кожної функції: розробка загальної моделі; складання списку необхідних функцій системи; планування роботи над кожною функцією; проектування функції; конструювання функції. Робота над проектом передбачає часті збори і ділиться на ітерації, кожна з яких реалізується за допомогою певного набору функцій.

Переваги FDD: робить процес легше; дає можливість планування і попереднього проектування, створення детального дизайну, є керування пріоритетами і можливе трасування вимог; розрахований на роботу у великих командах.

Але в той же час, недолік документації може значно збільшувати вартість подальшого супроводу продукту, оскільки внесення будь-яких змін до нього вимагатиме дуже великих зусиль.

4.4 Методологія Scrum

Методологія Scrum була вперше озвучена в роботі Хіротака Такеучи й Ікуджіро Нонаки, опублікованій в Harvard Business Review. Джеф Сазерленд використовував цю роботу при створенні методології для корпорації Easel у 1993 році, яку за аналогією і назвав Scrum, а Кен Швабер формалізував процес для використання в індустрії розроблення програмного забезпечення.

Мета цієї методології – виявити й усунути відхилення від бажаного результату на більш ранніх етапах розроблення програмного продукту.

Методологія Scrum визначає:

1. Правила, за якими повинен плануватися і управлятися список вимог до продукту, з метою досягнення максимальної прибутковості від реалізованої функціональності;
2. Правила планування ітерацій для забезпечення максимальної зацікавленості команди в отриманні результату;
3. Основні правила взаємодії учасників команди для максимально швидкої реакції на виникаючі робочі ситуації;
4. Правила аналізу і коригування процесу розроблення для вдосконалення взаємодії всередині команди.

Кожну ітерацію можна описати так: «Плануємо – Фіксуємо – Реалізуємо – Аналізуємо». За рахунок фіксування вимог на протязі однієї ітерації та зміни довжини ітерації можна керувати балансом між гнучкістю та плануємістю розроблення.

Scrum фокусується на постійному визначенні пріоритетних завдань, ґрунтуючись на бізнес цілях, що збільшує корисність і прибутковість проекту на його ранніх стадіях. Тому що при ініціації проекту його прибутковість визначити майже неможливо, Scrum пропонує концентруватися на якості розроблення і до кінця кожної ітерації мати проміжний продукт, який можна використовувати. Наприклад, результатом ітерації може бути каркас сайту, який можна показати на презентації.

Методологія Scrum орієнтована на те, щоб оперативно пристосовуватися до змін у вимогах, що дозволяє команді швидко адаптувати продукт до потреб замовника.

Девіз Scrum – «аналізуй та адаптуй»: аналізуйте те, що отримали, адаптуйте те, що вже розроблено до реальних потреб, а потім аналізуйте знову. Чим менше формалізму,

тим більш гнучко та ефективно можна працювати, – це основний принцип даної методології. Але це не означає, що формальних процесів не повинно бути зовсім, їх має бути достатньо для організації ефективної взаємодії і керування проектом. Формальна частина Scrum складається з трьох ролей, трьох практик і трьох основних документів.

В Scrum-проектах відділяють наступні ролі:

1. Власник продукту (Product Owner) – людина, що формулює вимоги програмістам. Зазвичай власник продукту є представником або довіреною особою замовника, а для компаній, що випускають коробкові продукти, він являє собою ринок, на якому реалізується продукт. Власник продукту повинен скласти бізнес план, який показує очікувану дохідність і план розвитку до вимог, відсортованими за коефіцієнтом окупності інвестицій. Виходячи з наявної інформації, власник продукту готує перелік вимог, відсортований за значимістю.

2. Scrum-майстер (Scrum Master) – забезпечує максимальну працездатність і продуктивність команди, чітку взаємодію між усіма учасниками проекту, своєчасне вирішення всіх проблем, що гальмують або зупиняють роботу будь-якого члена команди, відгороджує команду від усіх впливів ззовні під час ітерації та забезпечує проходження процесу всіх учасників проекту. Ця людина має бути одним з членів команди розроблення і брати участь у проекті як розробник. Він відповідає за своєчасне вирішення поточних проблем.

3. Scrum-команда (Scrum Team) – група, що складається з п'яти-дев'яти самостійних, ініціативних програмістів. Перше завдання цієї команди – поставити реально досяжну, прогнозовану, цікаву і значущу мету для ітерації. Друге завдання – зробити все для того, щоб ця мета була досягнута у відведені терміни і з заявленою якістю. Мета ітерації вважається досягнутою тільки в тому випадку, якщо всі поставлені завдання реалізовані, весь код написаний за певними проектом «стандартам кодування» (coding guidelines), програма протестована повністю, а всі знайдені дефекти усунені. Програмісти цієї команди повинні вміти оцінювати та планувати свою роботу, працювати в команді, постійно аналізувати та поліпшувати якість взаємодії та роботи. В обов'язки всіх членів Scrum-команди входить участь у виборі мети ітерації і визначення результату роботи. Вони повинні робити все можливе і неможливе для досягнення мети ітерації в рамках, визначених проектом, ефективно взаємодіяти з усіма учасниками команди,

самостійно організовувати свою роботу, надавати власнику робочий продукт в кінці кожного циклу. Scrum містить наступні документи: журнал продукту (Product Backlog), журнал спринту (Sprint Backlog) та графік спринту (Burndown Chart).

На початку проекту власник продукту готує журнал продукту (табл. 1.1) – перелік вимог, відсортований за значимістю, а Scrum- команда доповнює цей журнал оцінками вартості реалізації вимог. Перелік повинен містити функціональні та технічні вимоги, необхідні для реалізації продукту. Найбільш пріоритетні з них повинні бути досить детально прописані, щоб їх можна було оцінити і протестувати. Про своєчасноу деталізацію вимог має дбати власник продукту і надавати необхідний обсяг у потрібний час. У цьому сенсі програмісти є замовниками вимог для власника продукту. Надалі інші вимоги повинні поступово уточнюватися та деталізуватися до такого ж рівня. Головне, щоб у команди завжди був достатній обсяг підготовлених до реалізації вимог.

Таблиця 1.1 – Приклад журналу продукту

Номер вимоги	**Опис вимоги**	**Цінність бізнесу (ум. од.)**	**Пріоритет**	**Високорівнева оцінка (час)**
FE1	Покупець може зареєструватися на сайті	10.000	1	20
FE2	Покупець може ввести свої персональні дані	12.000	6	36
FE3	Покупець може побачити список доступних виробів	16.000	10	30
FE4	Покупець може купити виріб	100.000	20	48
FE5	Покупець може робити пошук виробів	80.000	30	32
FE6	Покупець може підписатись на новини	30.000	40	66

Після того як команда під час сесії планування вибрала і зобов'язалася реалізувати набір вимог з журналу продукту, ці вимоги розбиваються на більш дрібні завдання, що складають деталізований список вимог – журнал спринту (табл. 1.2). Розбивка на завдання повинна бути зроблена таким чином, щоб виконання однієї задачі займало не більше двох днів (вважається, що менш детальна, наприклад, півдня, розбивка призводить до більш грубої оцінки, ніж прийнятна в більшості проектів, що використовують методологію Scrum). Розбивка на завдання допоможе так спланувати ітерацію, щоб в кінці не залишилося жодної невиконаної завдання і, відповідно, досягти

її мети. Після завершення деталізації оцінка журналу спринту порівнюється з первинною оцінкою в журналі продукту. Якщо існує значна розбіжність, команда домовляється з власником продукту про обсяг робіт, який повинен бути виконаний протягом ітерації, і про те, який обсяг буде перенесено на наступну. Менш важливі і мало впливають на мету ітерації завдання виносяться з журналу спринту.

Графік спринту – показує щоденну зміну загального обсягу робіт, що залишився до закінчення ітерації. Цей графік дозволяє команді розробників робити аналіз поточної ситуації та своєчасно реагувати на відхилення.

Таблиця 1.2 – Приклад журнал спринту

№ вимоги	№ задачі	Опис вимоги	Пріо-ритет	Високо-рівнева оцінка (час)	Оцінка (час)	Зал роб (час)
FE1		Покупець може зареєструватися на сайті	1	20	24	19
	TA1	Розробити дизайн інтерфейсу користувача			1	0
	TA2	Закодувати інтерфейс Користувача			1	1
	TA3	Закодувати бізнес логіку та юніт тести			3	1
	TA4	Розробити структуру бази даних			1	1
	TA5	Закодувати таблиці та логіку бази даних			0,5	0,5
	TA6	Закодувати доступ до бази даних			2	2
	TA7	Задокументувати класи			1	1
	TA8	Написати тести для інтерфейсу користувача			2	1
	TA9	Написати тести функціональності			2	1
	TA10	Автоматизувати тести			2	2
	TA11	Зарезервовано на огляд коду			2	2
	TA12	Зарезервовано на виправлення дефектів			2	2
	TA13	Зарезервовано на спілкування з власником продукту			1	1
	TA14	Зарезервовано на рефакторинг			2	2
	TA15	Зарезервовано на самовдосконалення			1	1
	TA16	Зарезервовано на аналіз дефектів			0,5	0,5

Графік спринту дозволяє також власнику продукту спостерігати за ходом ітерації – якщо загальний обсяг робіт не зменшується щодня, значить, щось йде не так. Під час сесії планування команда знаходить і оцінює завдання, які треба виконати для успішного завершення ітерації. Сума оцінок всіх завдань в журналі спринту є загальним обсягом роботи, який треба виконати за ітерацію. Після завершення кожного завдання Scrum-майстер перераховує обсяг роботи, що залишилася, і зазначає це на графіку спринту. Тільки в тому випадку, якщо по закінченні ітерації у журналі спринту не залишилося незавершених завдань, ітерація вважається успішною. Графік спринту використовується як допоміжний інструмент, що дозволяє коригувати роботу для завершення ітерації вчасно, з працюючим кодом і необхідною якістю.

До Scrum-практик відносяться: спринт (Sprint), скрам (Daily Scrum Meeting) та демонстраційне засідання (Sprint Review Meeting).

1. Підготовка до першої ітерації, званої спринт (Sprint), починається після того, як власник продукту розробив журнал продукту.

При плануванні ітерації відбувається детальне розроблення сесій планування спринту (Sprint Planning Meeting), яке починається з того, що власник продукту, Scrum-команда і Scrum-майстер перевіряють план розвитку продукту, план релізу та перелік вимог, Scrum-команда перевіряє оцінки вимог, переконується, що вони досить точні, щоб почати працювати, вирішує, який обсяг роботи вона може успішно виконати за спринт, ґрунтуючись на розмірі команди, доступному часі та продуктивності. Scrum-команда повинна вибирати перші за пріоритетом вимоги з журналу продукту. Після того як Scrum-команда зобов'язується реалізувати обрані вимоги, Scrum- майстер починає планування спринту. Scrum-команда розбиває вибрані вимоги на завдання, необхідні для його реалізації. Ця активність в ідеалі не повинна займати більше чотирьох годин, і її результатом є журнал спринту. Необхідно, щоб всі учасники команди взяли на себе зобов'язання з реалізації обраної мети.

2. Після закінчення планування починається ітерація. Кожен день Scrum-майстер проводить «скрам» (Daily Scrum Meeting) – п'ятнадцятихвилинної нарада, мета якої – досягти розуміння того, що сталося з часу попередньої наради, скоригувати робочий план до реалій сьогодення і позначити шляхи вирішення існуючих проблем. Кожен учасник Scrum-команди відповідає на три питання: що я зробив з часу

попереднього скрама, що мене гальмує або зупиняє в роботі, що я буду робити до наступного скрама? У цій нараді може брати участь будь-яка зацікавлена особа, але тільки учасники Scrum- команди мають право приймати рішення. Правило обґрунтовано тим, що вони давали зобов'язання реалізувати мету ітерації, і тільки це дає впевненість у тому, що вона буде досягнута. 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.

В кінці кожного спринту проводиться демонстраційне засідання (Sprint Review Meeting) тривалістю не більше чотирьох годин. Спочатку Scrum-команда демонструє власнику продукту зроблену протягом спринту роботу, а той у свою чергу веде цю частину наради і може запросити до участі всіх зацікавлених осіб. Власник продукту визначає, які вимоги з журналу спринту були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети в журналі продукту для наступної ітерації. У другій частині мітингу проводиться аналіз минулого спринту, який веде Scrum-майстер. Scrum-команда шукає використані в останньому спринті позитивні і негативні способи спільної роботи, аналізує їх, робить висновки і приймає важливі для подальшої роботи рішення. Scrum-команда також визначає програми, які можуть працювати краще, і шукає шляхи для збільшення ефективності подальшої роботи. Потім цикл замикається, і починається планування наступного спринту.

Час між ітераціями – це час прийняття основоположних рішень, що впливають на хід всього проекту. Під час спринту ніякі зміни ззовні не можуть бути зроблені. Після того як команда дала зобов'язання реалізувати журнал спринту, він фіксується, і зміни в ньому можуть бути зроблені тільки з таких причин:

– Scrum-команда протягом ітерації отримала кращу уяву про вимоги і потребує додаткових завдань для успішного завершення ітерації;

– знайдені дефекти, які треба обов'язково виправити для успішного завершення ітерації;

– Scrum-майстер і Scrum-команда можуть вирішити, що невеликі зміни, які не впливають на загальний обсяг робіт, можуть бути реалізовані в зв'язку з виниклою у власника продукту необхідністю.

Виходячи з того що журнал спринту не може бути змінений ззовні під час ітерації, потрібно вибирати його довжину, ґрунтуючись на стабільності вимог. Якщо вимоги стабільні, змінюються або доповнюються рідко, то можна вибрати шеститижневий цикл. У цьому випадку заощаджується час на перемикання команди з активного розроблення на планування та демонстраційні мітинги. Якщо вимоги часто змінюються і доповнюються, потрібно відштовхуватися від двотижневого циклу, в будь-якому випадку довжина ітерації – це величина експериментальна.

Недоліки Scrum:

1. Складно домогтися активної участі від кожного розробника і злагодженої колективної роботи в команді;

2. Складно залучити постачальника вимог до активної участі в проекті, зацікавити його динамікою розвитку продукту, дати можливість бути активним вболівальником і спонсором команди.

Проте, незважаючи на це, використання методології Scrum в проектах дозволяє:

– використовувати весь технічний багаж, накопичений компанією, тому що головний напрям методології Scrum направлений на керування проектами і не задає ніяких технічних практик;

– з високою якістю та в рамках бюджету реалізувати великий обсяг функціональності та специфікації, які були відсутні на момент початку проекту;

– забезпечити максимальну бізнес-цінність виробленого продукту, за рахунок того, що практично всі реалізовані функції активно використовуються відвідувачами;

– досягнути високу супровідність коду (можливість внесення змін з мінімальними трудовитратами) – вартість змін, внесених до продукту, практично еквівалентна вартості розроблення аналогічних функцій продукту на початку проекту, що рідко буває у RUP чи MSF моделях виробництва, для яких характерний експонентний ріст вартості змін по мірі виконання проекту.

4.5 Методика Kanban

З початку 70-х років у Японії, а потім і в інших країнах набула великого поширення система «Канбан», що є механізмом організації безперервного гнучкого виробничого потоку і функціонує практично без страхових запасів. Традиційна концепція організації виробництва, як відомо, спрямована на запобігання простоям та організацію

безперервного потоку з обов'язковим створенням страхового запасу. Японська концепція ґрунтується на практично повній відмові від страхових запасів. Більше того, менеджери навмисне дають робітникам змогу повністю випробувати на собі наслідки простоїв. В результаті весь персонал постійно зайнятий виявленням причин збоїв у виробництві та пошуком шляхів підвищення надійності та запасу міцності системи управління. Після виявлення та усунення причин простоїв керівники ще більше скорочують страховий запас, породжуючи додаткові зусилля з покращення організації виробництва з боку всього персоналу. В умовах системи «Канбан», на відміну від традиційного підходу, виробник не має завершеного плану й графіка виробництва, а жорстко пов'язаний конкретним замовленням споживача. Він не взагалі оптимізує свою роботу, а в межах замовлення. Конкретного графіка роботи на декаду, місяць він не має. Кожен зайнятий у технологічному ланцюгу робітник знає, що він вироблятиме продукцію тільки тоді, коли карта «Канбан» з його продукції відкріплена від контейнера на складі, тобто коли продукція фактично надійшла на наступну стадію обробки. Конкретний графік послідовності праці одержують лінії кінцевого складання, вони розкручують клубок інформації у зворотному напрямі. Іншими словами, графіки виробництва не переглядаються, а тільки формуються рухом карток «Канбан», тому що зняття карти відбору продукції, графіка виготовлення її фактично не було.

Виробництво постійно перебуває в стані надбудови, відбувається його системне настроювання під зміни ринкової кон'юнктури.

Методика розробки програмного забезпечення Канбан була введена у практичне використання Девідом Андерсеном у 2007 році. Багато з цих практик та підходів використовувалися різними Agile командами, перш ніж були описані як єдине ціле.

Нововведення полягали в тому що ,було введено завдання "в процесі". Це робилося і раніше іншими Agile-командами, але в Канбан існує всім відоме обмеження на кількість робочих завдань, які можуть виконуватися в один час. Ця межа зазвичай досить низька – ліміт приблизно дорівнює числу розробників в команді або трохи менший.

Основні положення Канбан наступні:

1. Візуалізація потоку робіт:

1.1 Розбиття роботи на частини, кожна частина виписується на картку і прикріплюється до стіни;

1.2 Розбиття усіх завдань на стовпчики для розділення по стадіям розробки;

2. Обмеження незавершеної роботи (НЗР) (work-in-progress) визначається можлива кількість незавершених пунктів на кожній стадії робочого процесу

3. Вимірювання часу виконання завдання (lead time) (середньої тривалості часу для завершення одного пункту, іноді звану "оперативним часом" (cycle time)), оптимізація процесу, щоб звести час виконання завдання до мінімуму і зробити його настільки прогнозованим, наскільки це можливо.

Канбан – це не конкретний процес, а система цінностей. Його можна описати однією простою фразою – «Зменшення виконується в певний момент роботи (work in progress)».

Різниця між Канбан і Scrum:

– В Канбан немає таймбоксов (ні на завдання, ні на спринти);

– В Канбан завдання більше і їх менше;

– В Канбан оцінки термінів на задачу опціональні або взагалі їх немає;

– В Канбан «швидкість роботи команди» відсутній і вважається тільки середній час на повну реалізацію завдання.

Команда для роботи використовує Канбан-дошку. Наприклад, вона може виглядати так (стовпці зліва направо):

Цілі проекту. Сюди можна помістити високорівневі цілі проекту.

Черга завдань. Тут зберігаються завдання, які готові до того, щоб почати їх виконувати. Завжди для виконання береться зверху, сама пріоритетна задача і її картка переміщується в наступний стовпець.

Опрацювання дизайну. Цей та інші стовпці до «Закінчено» можуть змінюватися, тому що саме команда вирішує, які кроки проходить завдання до стану «Закінчено». Наприклад, в цьому стовпці можуть перебувати завдання, для яких дизайн коду або інтерфейсу ще не ясний і обговорюється. Коли обговорення закінчені, завдання пересувається в наступний стовпець.

Розробка. Тут завдання перебуває до тих пір, поки розробка не завершена. Після завершення вона пересувається в наступний стовпець. Або, якщо архітектура не вірна або не точна – завдання можна повернути в попередній стовпець.

Тестування. У цьому стовпці завдання знаходиться, поки воно тестується. Якщо знайдені помилки – повертається в Розробку. Якщо ні – пересувається далі.

Деплоймент. У всіх проектів різне поняття деплоймент, наприклад, викласти нову версію продукту на сервер або помістити код в репозиторій.

Закінчено. Сюди стікер потрапляє лише тоді, коли всі роботи по завданню закінчені повністю.

В будь-якій роботі трапляються термінові завдання. Для таких можна виділити спеціальне місце (наприклад «Прискорити»). В Прискорити можна помістити один рядок до завдання і команда повинна почати її виконувати негайно і завершити якомога швидше. У такій черзі може бути тільки одна така задача, якщо з'являється ще одна – вона повинна бути додана в «Черга завдань».

Під кожним стовпцем вказуються цифри, які позначають кількість завдань, які можуть бути одночасно в цих стовпцях. Цифри підбираються експериментально, але вважається, що вони повинні залежати від числа розробників в команді.

Наприклад, якщо є 8 програмістів в команді, то в рядок

«Розробка» можна помістити цифру 4. Це означає, що одночасно програмісти будуть робити не більше 4-х завдань, а значить у них буде багато причин для спілкування та обміну досвідом. Якщо поставити туди цифру 2, то 8 програмістів, що займаються двома завданнями, можуть простоювати або втрачати занадто багато часу на обговореннях. Якщо поставити 8, то кожен буде займатися своїм завданням і деякі завдання будуть затримуватися на дошці надовго, а головне завдання Канбан – це зменшення часу проходження завдання від початку до стадії готовності.

Використання дошки дає такі переваги:

1. Зменшення числа паралельно виконуваних завдань сильно зменшує час виконання кожної окремої задачі, за рахунок того, що немає необхідності перемикати контекст між завданнями, відстежувати різні сутності, планувати їх і т.д.

2. Легко побачити проблеми. Наприклад, якщо тестери не справляються з тестуванням, то вони дуже скоро заповнять весь свій стовпець і програмісти, які закінчили нову задачу, вже не зможуть перемістити її в стовпець тестування, тому що він заповнений. Для вирішення цієї проблеми, наприклад, програмісти можуть допомогти

тестерам завершити одну із завдань тестування і тільки тоді пересунути нове завдання на звільнене місце. Це дозволить виконати обидва завдання швидше.

3. Можна обчислити час на виконання усередненої задачі. Можна позначати на картці дату, коли вона потрапила в чергу завдань, потім дату, коли її взяли в роботу і дату, коли її завершили. За цим трьом точкам для хоча б 10 завдань можна порахувати середній час очікування в чергу завдань і середній час виконання завдання.

Підсумовуючи, переваги Канбан полягають у тому, що:

1. Легко визначаються проблемні області;

2. Зростає продуктивність роботи за рахунок того, що члени команди техпідтримки самостійно організують свою роботу, використовуючи канбан-дошку, а менеджер лише спрямовує зусилля на приорітезацію великих проектів і вирішення виникаючих проблем.

Недоліки Канбан:

1. Зі зменшенням НЗР з'являються обмеження – для менш пріоритетних проектів не вистачає ресурсів, що призводить до скорочення кількості проектів на команду.

2. Канбан накладає менше обмежень, ніж Scrum, тобто керівництво отримує більше параметрів для налаштування. Це може бути як недоліком, так і перевагою, залежно від ситуації.

3. Канбан – це ще більш «гнучка» методологія, ніж Scrum і XP, тому вона не підійде командам та проектам, не готовим до гнучкої роботи.