**Лабораторна робота №11.**  **Вибір, обгрунтування та застосування адаптивної методології для розробки ПЗ**

**Мета роботи**: засвоєння навичок визначення вибору, обгрунтування та застосування адаптивної методології для розробки ПЗ

**Завдання:**

1. Ознайомитися з теоретичною частиною, проаналізувати методології Scrum, Kanban, Scrumban, XP.
2. Для реалізації вашого проекту (можливо за умов розширення його функціональності) визначте:
3. кількість розробників для реалізації його протягом 2-3 місяців;
4. надайте аналіз методологій Scrum, Kanban, Scrumban, XP з точки зору їх застосування для розробки. Для цього заповніть таблиці:

|  |  |  |
| --- | --- | --- |
| Назва вашого проекту | | |
| Назва методології | Причини за яких ви її обираєте | Що необхідно для ефективної реалізації методології |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Назва вашого проекту | | |
| Назва методології | Причини за яких ви від неї відмовляєтесь | Що необхідно для застосування цієї методології в проекті |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. По закінченню лабораторну роботу потрібно здати на перевірку викладачеві, надіславши електронною поштою на адресу [t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com). Якщо викладач знаходить помилки чи неточності, він може повернути роботу на доопрацювання.

Файл з роботою повинен мати назву в такому форматі:

**KPZ <Номер групи><Номер лекції / лабораторної> [літера позначення типу роботи L – лекція, R – лабораторна]<Прізвище англійською>**.. Наприклад, **KPZ4101R**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

Тему в заголовку листа записати

**KPZ<Номер групи>-ЛР<Номер лабораторної>-<Прізвище >**

**Строк виконання цієї роботи ІПЗ-41 –**

**ІПЗ-42 – 09.11.2021**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, Тему в заголовку листа записати

**KPZ <Номер групи>-Запитання-<Прізвище >**.

**КОНТРОЛЬНІ ПИТАННЯ:**

1. Дайте визначення адаптивним методологіям.
2. Чи можливо використання прототипування за адаптивними методологіями?
3. В чому полягають особливості адаптивних методологій?
4. Для яких проектів доцільно застосовувати адаптивні методології?

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Потрібно виділити фактори, які можуть негативно вплинути на використання гнучких методів:

* масштабні зусилля у галузі розвитку (більше 20 розробників);
* розподілена у просторі команда;
* примусове впровадження гнучкого процесу усупереч вимогам команди розробників.

Небажане використання agile-методів у критично важливих системах (табл.1), де відмова ПЗ неможливий ні в якому разі (наприклад, управління повітряним рухом).

Таблиця 1 – Порівняння використання гнучких та формальних методів

|  |  |  |
| --- | --- | --- |
| Agile-методи | Методи із чітким планом | Формалізовані методи |
| Низька критичність | Висока критичність | Життєво важлива критичність |
| Досвідчені розробники | Молодші розробники | Досвідчені розробники |
| Вимоги часто змінюються | Вимоги змінюються не часто | Обмежені вимоги. |
| Малі команди розробників | Великі групи розробників | Вимоги обов’язково моделюються |
| Культура, що залежить від змін | Стала культура розроблення | Екстремальна увага на якості розроблення |

**Scrum**

Встановлює правила для процесу управління розробки ПЗ 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 ПЗ. 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.

**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 зворотного зв'язку

Організації що не створили другий рівень зворотного 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.

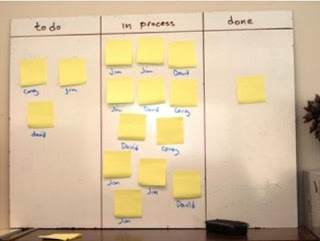
**Визначення Скраму.** Скрам – гнучкий фреймворк для вирішення складних адаптивних проблем. Скрам дуже мінімальний, простий для розуміння, але надзвичайно складний у використанні. Ролі, артефакти, події та правила Скраму не можуть змінюватися. Скрам є контейнером для інших процесів, технік та методологій, і може бути використаний у комбінації з ними. Дуже успішним є використання Скраму з інженерними практиками XP. «Скрам може бути використаний тільки цілком і функціонувати як контейнер для інших технік, методологій та практик» (ScrumGuide, 2011)

**Визначення Канбана.** Формальне визначення цього методу можна знайти в книзі Девіда Андерсона « [Kanban](https://translate.google.com/website?sl=ru&tl=uk&nui=1&prev=search&u=http://www.amazon.com/Kanban-ebook/dp/B0057H2M70/ref%3Dsr_1_1?ie%3DUTF8%26qid%3D1370762645%26sr%3D8-1%26keywords%3Dkanban) ». Метод складається лише з п'яти правил і не є методологією чи фреймворком. За визначенням автора «Канбан не є методологією. Також Канбан – це не спосіб розробки програмного забезпечення. Немає процесу Канбана для розробки програмного забезпечення. Принаймні я не знаю такого. Я ніколи про це не писав. Неможливо проводити розробку за допомогою одного Канбана. Метод Канбана сам собою не містить достатньо практик для продуктової розробки». (Дейвід Андерсон)

Ви можете працювати в «старому доброму» Водоспадному процесі та при цьому успішно користуватися Канбаном. Одне іншому не суперечить. За це на Канбан обрушується великий потік критики з боку Аджайл спільноти. Багато хто навіть не вважає його частиною Аджайла. Канбан є технікою обмеження роботи в процесі (WIP), яку можна «натягнути» на будь-який процес, що вже існує в команді.

Скрамбан є повноцінним Скрамом, усередині якого застосовується техніка Канбана. Один із головних ідеологів Канбана та Ліна – Корі Ладас видав чудову книгу «Scrumban». У ній він вказав на слабкі місця Скраму і показав, яким чином, додавши Канбан, можна підвищити продуктивність команди та оптимізувати потік усередині Спринту. Формула Скрамбана наступна:  **Scrumban = Scrum + Kanban.**  У команди має бути впроваджений повнофункціональний Скрам та на додаток «навішений» Канбан. Таке поєднання є дуже ефективним. Типова СкрамБан команда використовує повноцінний Скрам та дошку для візуалізації своєї роботи з явно проставленими обмеженнями на роботу (інакше це не Канбан) у прогресі.

Уявімо, що Скрам команда почала працювати над ВСІМИ елементами Беклога Спринта в перший же день відразу після планування. Інакше кажучи, всі елементи Спринт Беклога «поїхали» одночасно. Формально це жодним чином не порушує правила Скраму. Справді, Скрам не пояснює, як команда розробки повинна доставляти обраний на плануванні функціонал. Спринт – це чорна скринька, в якій не видно, що відбувається. На вході такої скриньки – Спринт Беклог та Мета Спринту, а на виході – Інкремент продукту (Potentially Shippable Product Increment).

[](https://translate.google.com/website?sl=ru&tl=uk&nui=1&prev=search&u=http://1.bp.blogspot.com/--b6kHawpN5M/Ufoqi7LXoUI/AAAAAAAABOw/pdq861iDgKk/s1600/scrumWIP.JPG)

Природно, що мати таку велику кількість роботи в прогресі не є оптимальним підходом. Це одна з багатьох причин, чому багато Скрам команди зі Спринту до Спринту можуть не приносити той обсяг функціоналу, який вони прогнозують доставити на плануванні.

СкрамБан команда ефективно вирішує цю проблему тим, що обмежує роботу у прогресі. Дошка перетворюється і виглядає так:

[](https://translate.google.com/website?sl=ru&tl=uk&nui=1&prev=search&u=http://2.bp.blogspot.com/-ZpUVKWimEZ8/UfoqcZnqSiI/AAAAAAAABOo/ZP5X33uVTOE/s1600/scrumban1.JPG)

**Немає поганих інструментів, є невірний контекст їхнього використання**

Використання Скрама та Канбана дуже ефективно. За допомогою Скраму ми вирішуємо наші стратегічні завдання та працюємо, використовуючи цикли зворотного зв'язку та прозорість. За допомогою Канбана вирішуємо тактичні завдання усередині Спринту та вирівнюємо потік.

Як стверджується у біблії управління процесами (“ProcessDynamics, Modeling, andControl“, 1994), поганих процесів немає. Проте іноді процеси застосовуються в невірному контексті. Скрам – емпіричний процес, побудований за принципами Бережливого Виробництва. Найбільше він підходить для вирішення комплексних складних завдань і тому дуже успішний у галузі розробки програмного забезпечення, де більше невідомого та невизначеного.

“Скрам, Канбан та XP – доповнюючі один одного, конкуруючі та часто конфліктуючі моделі. Але це не є великою проблемою. Просто ми маємо бути обережними та досить критичними у використанні цих моделей та знань, які вони нам дають” (Юрген Апелло, Management 3.0).

стання методології RUP.

**Методологія eXtreme Programming (XP)**

Екстремальне програмування (eXtreme Programming, XP) – спрощена методологія організації виробництва для невеликих і середніх за розміром команд розробників, які займаються розробленням програмного продукту в умовах незрозумілих або швидко змінних вимог.

Програмування відповідно до методик ХР доводить використання загальноприйнятих принципів програмування до екстремальних рівнів:

перегляд коду виконується постійно (з урахуванням того, що програмування ведеться парами);

* кожен учасник проекту тестує код програми постійно (тестування модулів), навіть замовники проводять функціональне тестування;
* проектування є складовою частиною повсякденної роботи кожного розробника (перероблення коду);
* розроблення виконується з урахуванням вимоги збереження в системі найбільш простого дизайну, що забезпечує поточний необхідний рівень функціональності (простіші речі надійніші в роботі);
* увага до архітектури системи на кожному етапі проекту;
* інтеграційне тестування виконується після кожної найменшої зміни у системі (триваюча інтеграція);
* ітерації невеликі – тривають години, кілька днів (постійне планування).

У розділі конспекту «Керування та організація робіт» наведені основні ризики розроблення ПЗ, виділені К.Беком. Для зменшення кількості можливих ризиків та усунення їх негативного впливу і розроблена методологія XP. У табл.2 наведені найпоширеніші ризики розроблення та методи їх усунення за допомогою екстремального програмування.

Таблиця 2 – Методи усунення ризиків розроблення ПЗ у методології XP

| Ризик | Метод подолання у XP |
| --- | --- |
| Зміна графіка | Планується короткострокове розроблення версій ПЗ |
| Закриття проекту | Замовник визначає найменший перелік найважливіших функцій системи, які визначають якість системи |
| Система втрачає корисність | Постійне розроблення та виконання тестів над системою не дозволяють накопичуватися дефектам, що впливають на роботу системи |
| Велика кількість дефектів і недоліків | Тести постійно створюються і виконуються не лише розробниками, а й замовниками, які перевіряють функціональність системи |
| Невідповідність ПЗ розв'язуваній проблемі | Представник замовника є постійним членом команди розробників. Специфікація постйно перевіряється і змінюється за потреби |
| Зміна характеру бізнесу | Бізнес не встигає помітно змінити напрямок діяльності у ході короткострокового розроблення версії програми |
| Нестача функціональних можливостей | Задачі із найбільшим пріоритетом реалізуються у першу чергу |
| Плинність кадрів | Програмісти самостійно оцінюють обсяг робіт та термін виконання, що підвищує їх зацікавленість |

Зміна графіка робіт є найпоширенішою проблемою під час створення ПЗ. Неуважність до розтягування робіт може зірвати увесь проект і стати причиною усіх інших ризиків. На кожну версію програми при використанні XP виділяється один-два місяці. У рамках кожної версії планується кілька ітерацій для отримання запланованої функціональності, кожна з яких триває один-чотири тижні. Замовник обов’язково перевіряє функціональність, отриману в ході чергової ітерації, тобто постійно забезпечується зв'язок із замовником, що отримує уявлення про поточний стан робіт. Кожна ітерація розділяється на кілька задач, які тривають кілька днів. Найважливіші завдання реалізуються в першу чергу. Тобто при зміщенні строків робіт можна бути впевненим, що не робробленими залишилися завдання низького пріоритету. Використання коротких етапів проекту для розроблення чергової версії програми дозволяє гнучко керувати строками робіт.

Для формування стилю розроблення, що дозволить досягти потрібної якості рішення, в XP пронується керуватись чотирма цінностями:

* комунікацією (communication) – дисципліна XP, спрямована на забезпечення безперервної комунікації усіх учасників проекту. Під час тестування, програмування в парах та попереднього оцінювання робіт замовники, розробники та менеджери змушені постійно спілкуватись;
* простотою (simplicity) – неможливо заздалегідь точно визначити, як буде розвиватися проект, тому вирішувати необхідно лише завдання, що виникають сьогодні, у найбільш простий спосіб;
* зворотним зв'язком (feedback) – спосіб отримати точні та конкретні дані про стан проекту. Постійне виконання тестів для перевірки усіх змін у системі забезпечує програміста звороним зв’язком про якість роботи. Кожен новий опис замовником вимог до системи одразу ж оцінюється розробниками і забезпечує замовника зворотним зв’язком із інформацію про якість опису. Менеджер, який контролює строки виконання робіт, забезпечує усіх учісників проекту інформацією про виконання планових термнів розроблення. В XP найбільш важливі функції системи реалізуються в першу чергу у реально працюючий продукт, тому замовник досить швидко може оцінити хід виконання та якість розроблення, а також відповідність замовленню;
* хоробрість (courage) – для того, щоб розроблення не втратила своєї актуальності, в XP рішення повинні прийматися з максимальною швидкістю, для чого потрібна певна сміливість.

Для визначення методів вирішення проблем розроблення ПЗ на основі обраних цінностей потрібно керуватися такими фундаментальними принципами оцінки методів вирішення поставленого завдання:

* швидкий зворотний зв'язок – інформація про стан системи повинна якомога швидше надходити до зацікавлених сторін проекту. У рамках дисципліни XP ці дані повинні із максимальною швидкістю надходити, інтерпретуватися та на основі їх аналізу швидко виконуватися модифікації системи;
* прийнятна простота – кожна проблема повинна вирішуватись у найбільш простий спосіб. У рамках XP значні зусилля (тестування, переробка коду, комунікації) покладаються для вирішення завдань сьогодення таким чином, щоб завтра внесення змін не потребувало великих зусиль;
* поступова зміна – кардинальні зміни часто приводять до провалу проекту, щоб заплановані модифікації мали успіх, їх потрібно реалізовувати як серію невеликих змін, після кожної з яких перевіряється працездатність системи;
* прийнятна зміна – найвигідніша стратегія – та, що дозволяє вирішити найбільш важливу проблему і залишає максимальну свободу дій;
* якісна робота – кожен учасник проекту повинен прагнути максимально якісно виконувати свої завдання і сприяти якісній роботі інших.

Кожен принцип втілює заявлені цінності. З-поміж альтернативних методів, які втілюють принципи, в XP пропонується обирати метод рішення, який відповідає виконанню більшості принципів.

У рамках XP пропонуються методи для виконання чотирьох основних видів діяльності у ході розроблення ПЗ:

* Кодування – від якості коду залежать робота та надійність системи. Якісне кодування потребує постійного навчання та удосконалення. Також код містить інформацію про стан проекту у найльш стислій та чітко зрозумілій формі;
* Тестування – постійне тестування зменшує вартість внесення змін у розроблення та зменшує ризики, пов’язані із змінами. Виконання усіх запланованих тестів свідчить про успішне завершення чергової версії або усього проекту;
* Слухання – відкрите чесне спілкування – як між розробниками, так і між розробниками і замовниками. Програмісти повинні почути бізнес-вимоги замовника, щоб виконати завдання, а замовник повинен чути зауваження розробників, щоб краще розуміти свої потреби;
* Проектування – проект системи впливає на якість ПЗ і на легкість внесення змін. Правильний проект створює структуру, що організовує логіку системи. Логічно пов’язані фрагменти ПЗ повинні поєднуватись у незалежні частини системи. При правильному дизайні програми внесення змін в одному елементі системи не буде потребувати змін у інших.

Для розв’язання виділених проблем розроблення в методології XP використовуються такі методики:

* Гра в планування (planning game) – швидко визначає обсяг робіт, що необхідно виконати у наступній версії. Для цього розглядаються бізнес-пріоритети та технічні оцінки. Якщо з часом план перестає відповідати дійсності, відбувається оновлення плану. Замовники повинні приймати рішення стосовно обсягів робіт, пріоритетності завданть та строків випуску версій. Розробники повинні відповідати за оцінку часу, потрібного на реалізацію вимог, наслідки прийнятих рішень, організацію процесу розроблення та детальне планування робіт.
* Невеликі версії (small releases) – найперша спрощена версія системи, що реалізує найбільш важливу функціональність, швидко вводиться в експлуатацію. Наступні версії випускаються через відносно короткі проміжки часу.
* Метафора (metaphor) – проста загальнодоступна і загальновідома історія, що коротко описує, як працює вся система. Ця історія керує усім процесом розроблення.
* Простий дизайн (simple design) – у кожен момент часу система повинна бути спроектована так просто, як це можливо. Виявлена надмірна складність усувається одразу.
* Тестування (testing) – програмісти постійно пишуть тести для модулів, а замовники – тести, які демонструють працездатність і завершеність тієї чи іншої можливості системи. Умова продовження розроблення – усі тести спрацьовують без помилок.
* Переробка (refactoring) – програмісти реструктурують систему, не змінюючи її поведінки. При цьому вони усувають дублювання коду, покращують комунікацію, спрощують код і підвищують його гнучкість.
* Програмування парами (pair programming) – увесь розроблювальний код пишеться двома програмістами на одному комп'ютері.
* Колективне володіння (collective ownership) – у будь-що момент часу будь-що член команди може змінити будь-що код у будь-якому місці системи.
* Безперервна інтеграція (continuous integration) – система інтегрується і збирається кожного разу, коли завершується рішення чергового завдання (можливо кілька разів за день).
* 40-годинний тиждень (40-hour week) – програмісти працюють не більше 40 годин на тиждень. Ніколи не можна працювати понаднормово два тижні поспіль.
* Замовник на місці розроблення (on-site customer) – до складу команди входить реальний користувач системи. Він доступний упродовж усього робочого дня і здатний відповідати на запитання про систему.
* Стандарти кодування (coding standards) – програмісти пишуть увесь код у відповідності до правил, які забезпечують комунікацію за допомогою коду.