Lesson 26

01.03.2024

```
public class Test1 {
    public static void main(String[] args) {
        byte a = 12;
        byte b = 13;
        byte result = a + b;
        System.out.println(result);
    }
}
```

```
public class Test2 {
    public static void main(String[] args) {
        int x = 011;
        int y = 0xfee;
        int result = x + y;
        System.out.println(result);
    }
}
```

```
public class Test3 {
    public static void main(String[] args) {
        int i = 0;
        int j = 9;
        do {
            i++;
            if (j-- < i++) {
                break;
            }
        } while (i < 5);
        System.out.println(Integer.toString(i)
                + Integer.toString(j));
    }
}
```

```
public class Test4 {
    public static void main(String[] args) {
        int[] x = new int[3];
        System.out.println("x[0] is " + x[0]);
    }
}
```

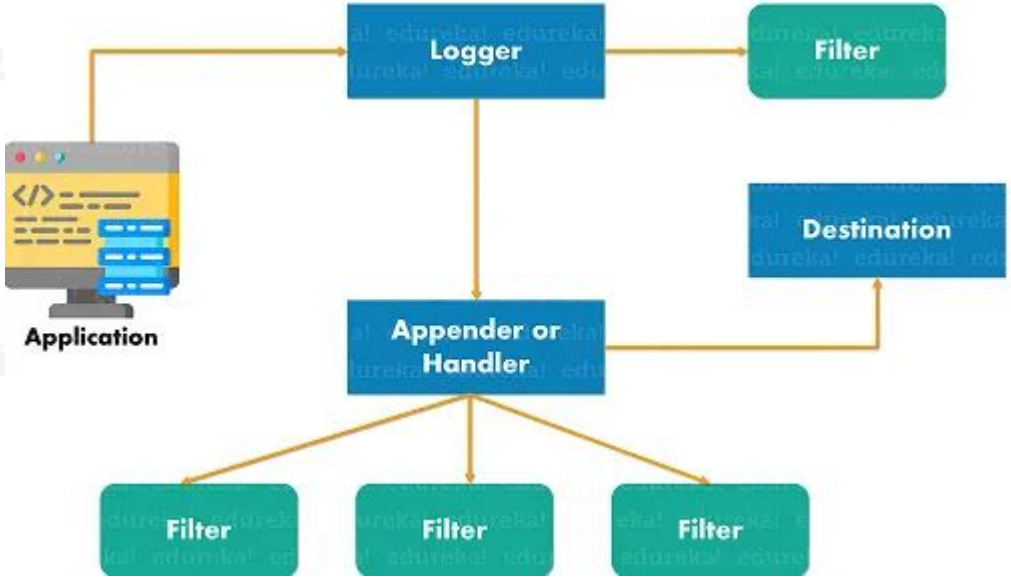
```
public class Test5 {
    public static void main(String[] args) {
        int x = 5;
        int y = 6;
        if ((y = 1) == x)
            System.out.println(y);
        else
            System.out.println(++y);
    }
}
```

```
public class Test6 {
    public static void main(String[] args) {
        for (int i = 0; i < 3; ) {
            System.out.println("Java");
        }
    }
}
```


5 java questions

Java™

Logging API



Loggers — відповідають за захоплення записів журналу та передачу їх відповідному Appender.

Appenders or Handlers — вони відповідають за запис подій журналу до місця призначення. Додатки форматують події за допомогою Layouts перед надсиланням виходів

Layouts or Formatters — відповідальні за визначення того, як виглядають дані, коли вони з'являються в записі журналу.

DEBUG fine-grained informational events that are most useful to debug an application

INFO informational messages that highlight the progress of the application at coarse-grained level

WARN potentially harmful situations

ERROR error events that might still allow the application to continue running

FATAL very severe error events that will presumably lead the application to abort

Logging Level	Detail Included in Logs				
	Debug statements	Informational messages	Warning statements	Error statements	Fatal statements
ALL	Included	Included	Included	Included	Included
DEBUG	Included	Included	Included	Included	Included
INFO	Excluded	Included	Included	Included	Included
WARN	Excluded	Excluded	Included	Included	Included
ERROR	Excluded	Excluded	Excluded	Included	Included
FATAL	Excluded	Excluded	Excluded	Excluded	Included
OFF	Excluded	Excluded	Excluded	Excluded	Excluded

slf4j	Log4j	Log4j2	Logback	java.util.logging
FATAL	FATAL	FATAL		
ERROR	ERROR	ERROR	ERROR	SEVERE
WARN	WARN	WARN	WARN	WARNING
DEBUG	INFO	INFO	DEBUG	INFO
INFO	DEBUG	DEBUG	INFO	CONFIG
				FINE
TRACE	TRACE	TRACE	TRACE	FINER
				FINEST

Розглянемо рівні на прикладі log4j, тут вони в порядку спадання:

FATAL: помилка, після якої програма більше не зможе працювати і буде зупинена, наприклад, помилка JVM бракує пам'яті;

ERROR: рівень помилок, коли є проблеми, які потрібно вирішити. Помилка не зупиняє роботу програми в цілому. Інші запити можуть працювати правильно;

WARN: об'явлений логі, який містить попередження. Сталася неочікувана дія, хоча система продовжувала працювати та виконала запит;

INFO: журнал, який записує важливі дії в додатку. Це не помилка, це не попередження, це очікувана дія системи;

DEBUG: журнали, необхідні для налагодження програми. Для впевненості, що система робить саме те, що від неї очікується, або опис дій системи: «метод1 почав роботу»;

TRACE: менш пріоритетні журнали для налагодження з найнижчим рівнем журналювання;

ALL: рівень, на якому будуть записані всі журнали з системи.

Що потрібно логувати

Зрозуміло, логувати все поспіль не варто. Іноді це не потрібно, і навіть небезпечно. Наприклад, якщо залогувати чиїсь особисті дані і це якимось чином спливе на поверхню, то будуть реальні проблеми, особливо на проектах, орієнтованих на Захід. Але є й те, що обов'язково логувати:

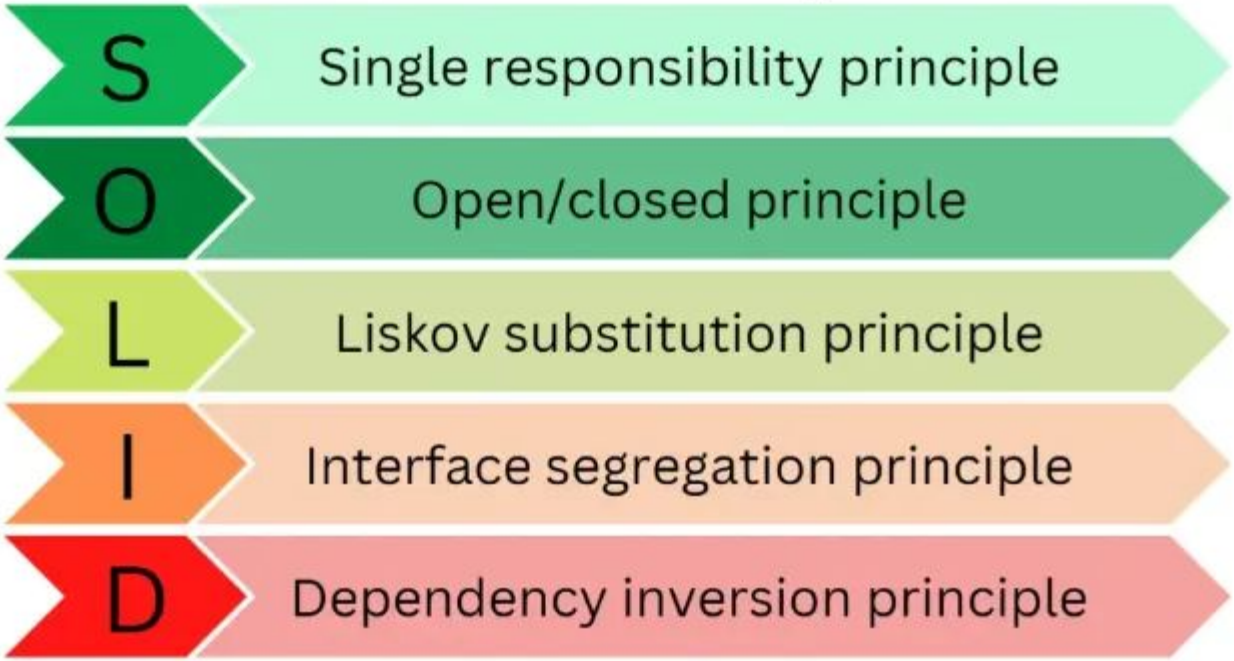
- Початок/кінець роботи програми. Потрібно знати, що програма дійсно запустилася, як ми і очікували, і завершилася так само очікувано.
- Питання безпеки. Тут добре б логувати спроби підбору пароля, логування входу важливих користувачів тощо.
- Деякі стани програми. Наприклад, перехід із одного стану в інший у бізнес процесі.
- Деяка інформація для дебага, з відповідним рівнем логування.
- Деякі SQL скрипти. Є реальні випадки, коли це потрібне. Знов-таки, вмілим чином регулюючи рівні, можна досягти відмінних результатів.
- Нитки (Thread), що виконуються, можуть бути логовані у випадках з перевіркою коректної роботи.

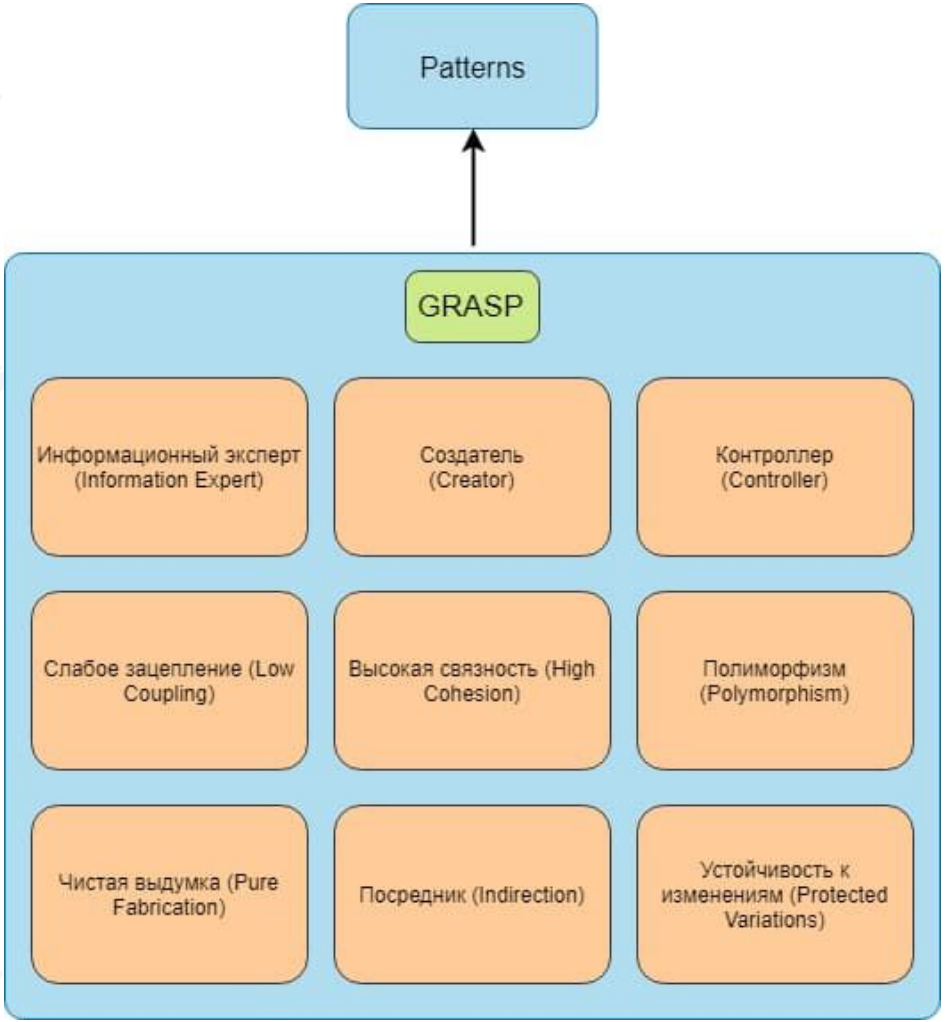
Популярні помилки у логуванні

Нюансів багато, але можна виділити кілька частих помилок:

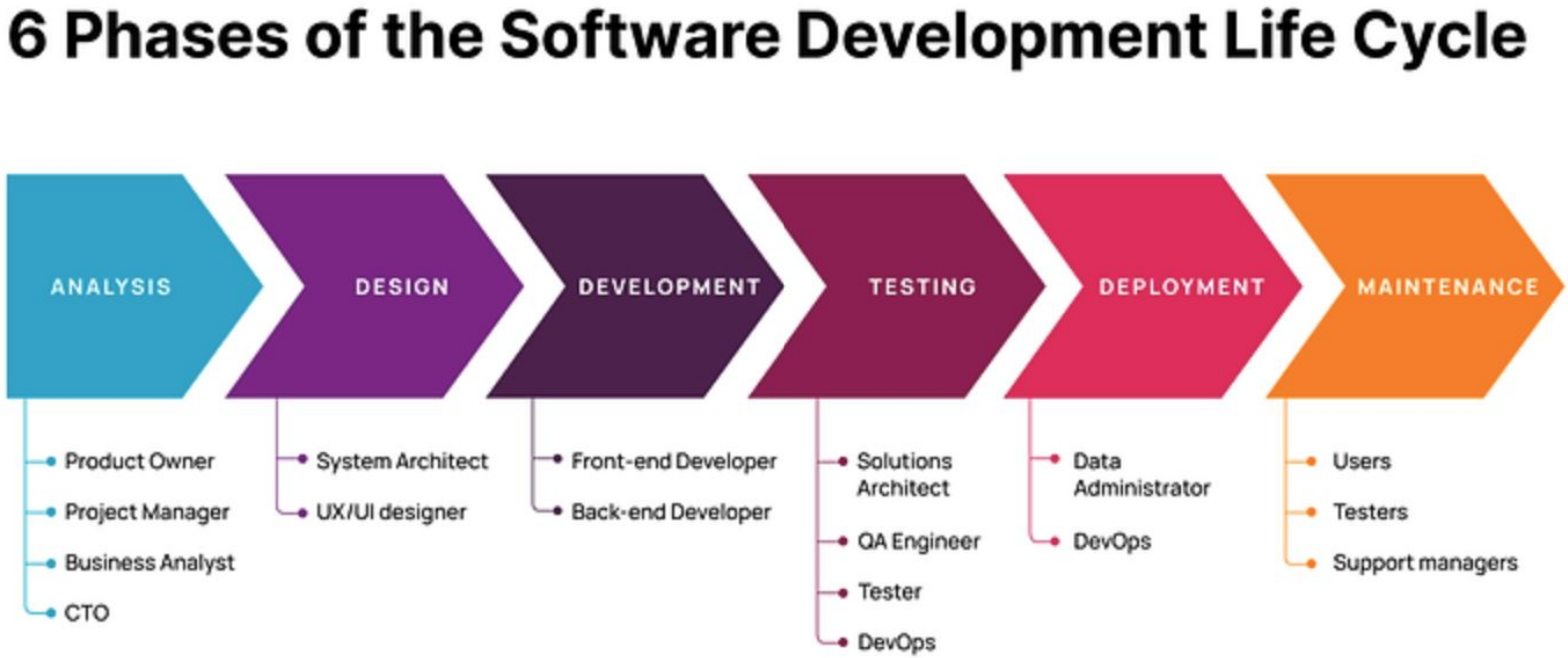
- Надлишок логування. Не варто логувати кожен крок, який суто теоретично може бути важливим. Є правило: логи можуть навантажувати працездатність трохи більше, ніж 10%. Інакше будуть проблеми із продуктивністю.
- Логування всіх даних на один файл. Це призведе до того, що в певний момент читання/запис у нього буде дуже складним, не кажучи про те, що є обмеження за розміром файлів у певних системах.
- Використання неправильних рівнів логування. Кожен рівень логування має чіткі межі, і їх варто дотримуватися. Якщо межа розпливчаста, можна домовитися який із рівнів використовувати.

SOLID Principles




Patterns
GRASP
Информационный эксперт (Information Expert)
Создатель (Creator)
Контроллер (Controller)
Слабое зацепление (Low Coupling)
Высокая связность (High Cohesion)
Полиморфизм (Polymorphism)
Чистая выдумка (Pure Fabrication)
Посредник (Indirection)
Устойчивость к изменениям (Protected Variations)

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



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 девелопером, перевіряється іншим. Також дуже часто використовують парне програмування, де один кодер пише код, а його партнер цей код одразу перевіряє.

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.

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.

Lean software development

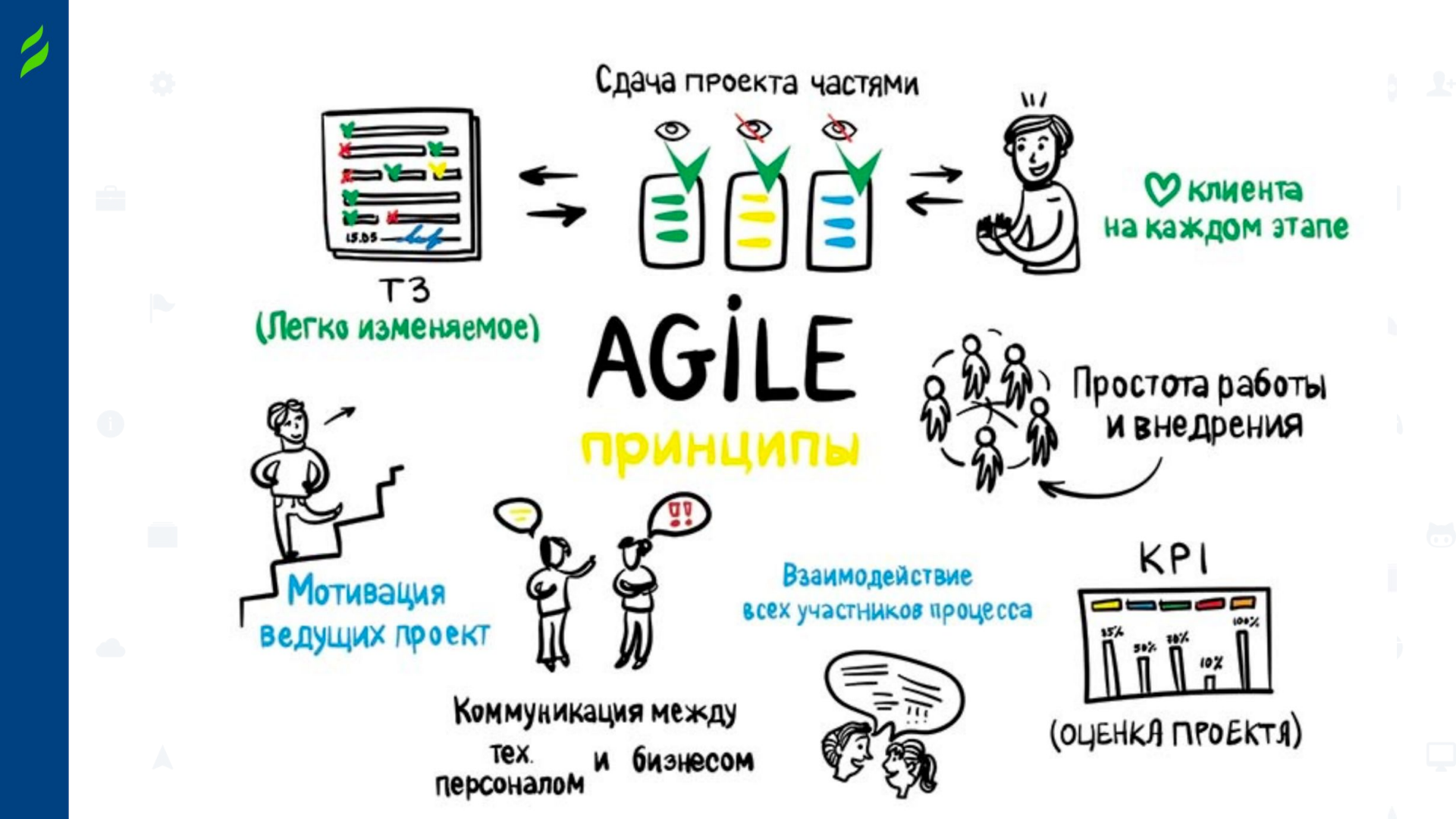
Ця методологія розробки програмного забезпечення метод бережливого виробництва (Lean Manufacturing). Основні принципи цієї методології є: максимальна швидка доставка продукції замовнику з дуже короткими ітераціями, мотивація команди і зосередитися на навчанні з короткими циклами розвитку, раннє тестування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

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.

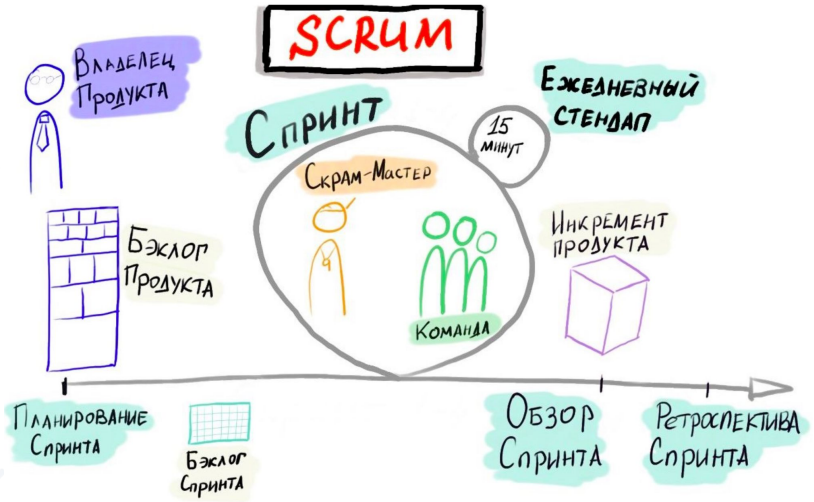
Scrumban

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.


Сдача проекта частями
ТЗ
(Легко изменяемое)
клиента
на каждом этапе
AGILE
принципы
Простота работы
и внедрения
Мотивация
ведущих проект
Взаимодействие
всех участников процесса
KPI
(ОЦЕНКА ПРОЕКТЯ)
Коммуникация между
тех. персоналом
и бизнесом

Методологія "Scrum"

Scrum - гнучкий метод управління проектами. В його основі лежать «спринти» -короткі ітерації, суворо обмежені в часі (зазвичай, 2-4 тижня). Тривалість нарад при цьому зводиться до мінімуму, але збільшується їхня частота. Кожен спринт складається зі списку завдань до кінця ітерації, і кожна операція має свою “вагу”. Під час нарад команда обговорює, хто що зробив, що збирається зробити і які проблеми. Для планування у Scrum використовують журнал спринтів. У такому підході в команді часто з'являється Scrum майстер, який налагоджує безперервну роботу всієї команди, створюючи для її комфортні умови. Також на проекті з'являється роль Product Owner. керівника розробки, людини, яка стежить за продуктом та виступає головною ланкою між запитом клієнта та результатом команди



Методологія «Kanban»

Найважливішою особливістю Kanban є візуалізація життєвого циклу проекту. Створюються стовпчики виконання завдань, які здаються індивідуально. Колонки позначені маркерами типу: To do, In progress, Code review, In testing, Done (назва колонок, звичайно ж, може змінюватися). Мета кожного учасника команди - зменшувати кількість завдань у першій колонці. Підхід Kanban наочний і допомагає зрозуміти де виникла проблема. Структуру Kanban не визначають остаточно і безповоротно: залежно від специфіки проекту можна додати імпровізовані колонки. Наприклад, деякі команди використовують систему, у якій потрібно визначити критерії готовності завдання перед виконанням. Тоді додають дві колонки – specify (уточнити параметри) та execute (взятися за роботу)