Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні системи та технології

# Лабораторна робота №2

# **Технології розроблення програмного забезпечення**

*Діаграма варінтів використання. Сценарії варінтів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи*

Варіант 12 (CI server)

Виконав
студент групи ІА-13:

Матусяк М.Ю.

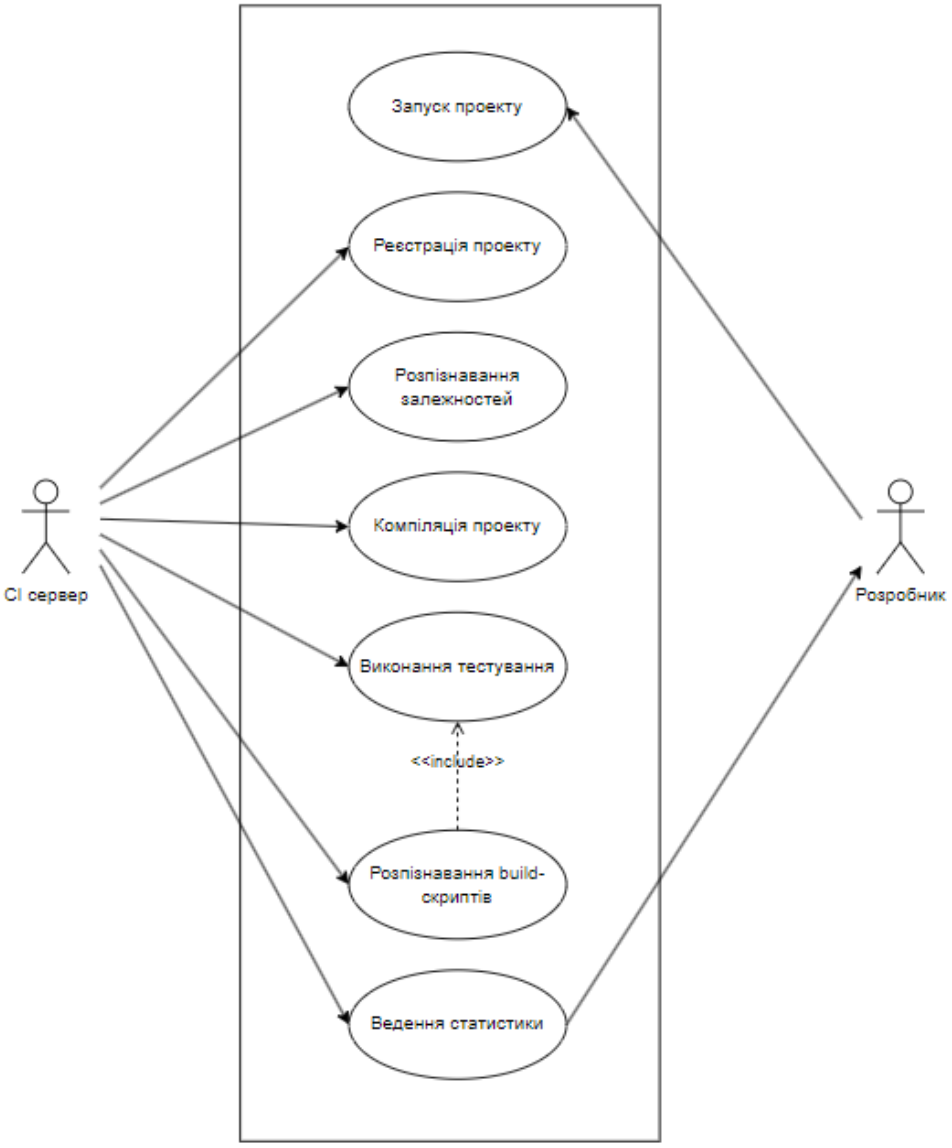
Перевірив

Драган М.С.

Київ 2023

**Мета роботи:** навчитися створювати діаграму варіантів використання, сценарії варіантів використання, UML діаграми, діаграми класів, концептуальну модель системи.

**Хід роботи:**




Діаграма прецедентів

Прецедент 1: Запуск проекту

*Опис:* Розробник ініціює запуск проекту для перебудови та тестування.

*Актори:* Розробник

*Умови успішності:* Розробник має доступ до системи CI Server.

*Основний сценарій:*

1. Розробник входить до системи CI Server.
2. Розробник обирає проект, який він бажає запустити.
3. Розробник ініціює запуск проекту.

4. CI Server розпочинає перебудову та тестування обраного проекту.
5. Після завершення процесу, CI Server повідомляє розробника про результати (успішно або невдалий запуск).

Прецедент 2: Реєстрація нового проекту

*Опис:* CI Сервер реєструє новий проект для подальшої перебудови та тестування.
*Актори:* CI Сервер
*Умови успішності:* CI Сервер працює та доступний для реєстрації проектів.

*Основний сценарій:*

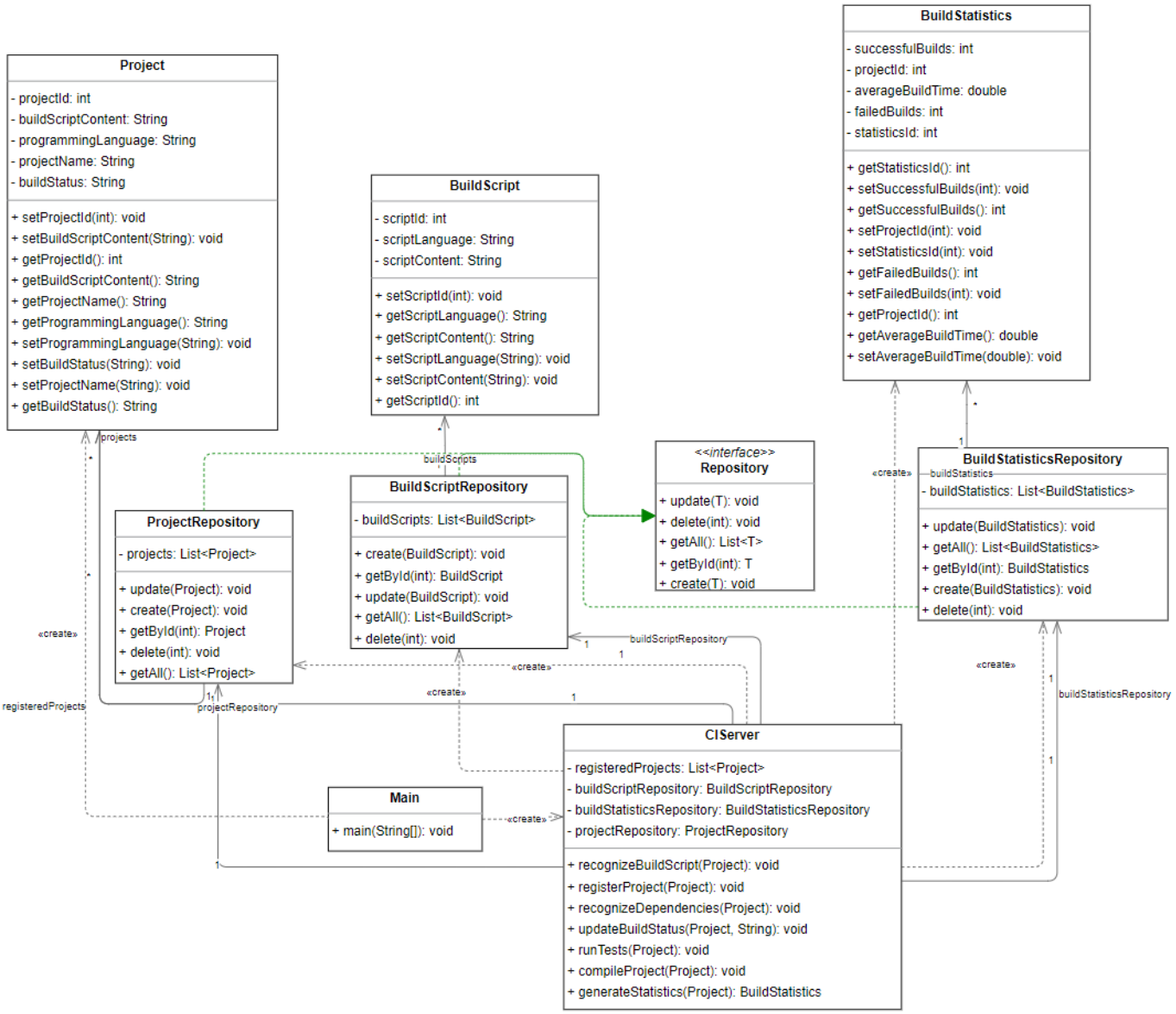
1. CI Сервер очікує на запити про реєстрацію нового проекту.
2. Розробник відправляє запит на реєстрацію нового проекту, надаючи інформацію про проект, його мову програмування та залежності.
3. CI Сервер реєструє новий проект у своїй системі та налаштовує його для подальшої збірки та тестування.
4. CI Сервер надсилає підтвердження реєстрації розробнику.

Прецедент 3: Відслідковування статистики збірок

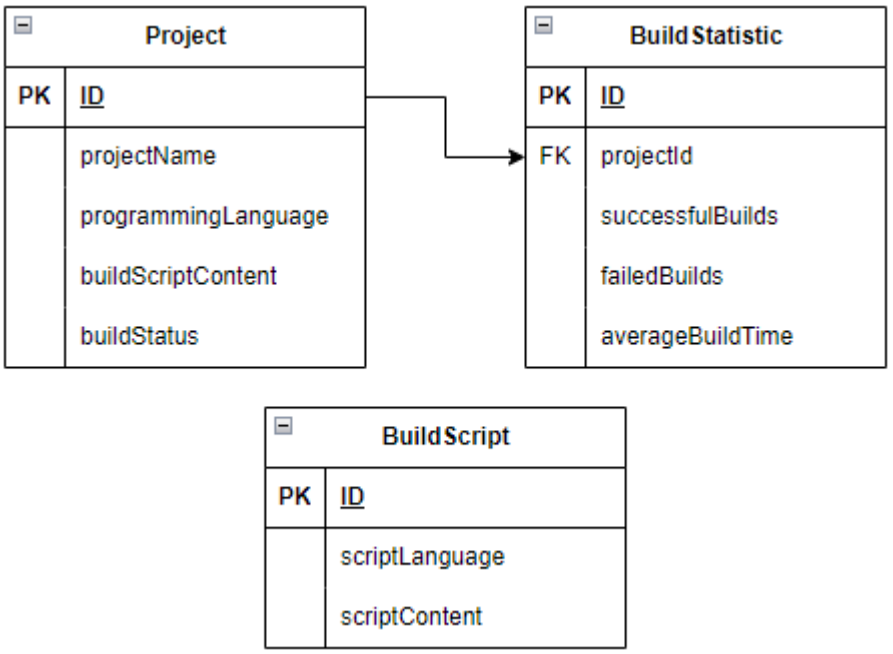
*Опис:* Розробник або адміністратор системи може відслідковувати статистику збірок для проектів на CI сервері.
*Актори:* Розробник або адміністратор системи
*Умови успішності:* Розробник або адміністратор мають доступ до системи CI Server.

*Основний сценарій:*

1. Розробник або адміністратор входить до системи CI Server.
2. Вони обирають проект, для якого бажають переглянути статистику збірок.
3. CI Server надає інформацію про кількість успішних та невдалих збірок для обраного проекту.
4. Вони можуть також переглядати середній час збірки та інші важливі метрики для оцінки ефективності проекту на CI сервері.




Діаграма класів системи




Структура бази даних

Вихідні коди класів системи:

```java
public class Project {
    2 usages
    private int projectId;
    2 usages
    private String projectName;
    2 usages
    private String programmingLanguage;
    2 usages
    private String buildScriptContent;
    2 usages
    private String buildStatus;

    no usages
    public int getProjectId() { return projectId; }

    no usages
    public void setProjectId(int projectId) { this.projectId = projectId; }

    no usages
    public String getProjectName() { return projectName; }

    no usages
    public void setProjectName(String projectName) { this.projectName = projectName; }

    no usages
    public String getProgrammingLanguage() { return programmingLanguage; }

    no usages
    public void setProgrammingLanguage(String programmingLanguage) { this.programmingLanguage = programmingLanguage; }

    no usages
    public String getBuildScriptContent() { return buildScriptContent; }

    no usages
    public void setBuildScriptContent(String buildScriptContent) { this.buildScriptContent = buildScriptContent; }

    no usages
    public String getBuildStatus() { return buildStatus; }

    1 usage
    public void setBuildStatus(String buildStatus) { this.buildStatus = buildStatus; }

}
```

```java
public class BuildScript {
    2 usages
    private int scriptId;
    2 usages
    private String scriptLanguage;
    2 usages
    private String scriptContent;

    no usages
    public int getScriptId() {
        return scriptId;
    }

    no usages
    public void setScriptId(int scriptId) { this.scriptId = scriptId; }

    no usages
    public String getScriptLanguage() { return scriptLanguage; }

    no usages
    public void setScriptLanguage(String scriptLanguage) { this.scriptLanguage = scriptLanguage; }

    no usages
    public String getScriptContent() { return scriptContent; }

    no usages
    public void setScriptContent(String scriptContent) { this.scriptContent = scriptContent; }
}
```

```java
public class BuildStatistics {
    2 usages
    private int statisticsId;
    2 usages
    private int projectId;
    2 usages
    private int successfulBuilds;
    2 usages
    private int failedBuilds;
    2 usages
    private double averageBuildTime;

    no usages
    public int getStatisticsId() {
        return statisticsId;
    }

    no usages
    public void setStatisticsId(int statisticsId) { this.statisticsId = statisticsId; }

    no usages
    public int getProjectId() { return projectId; }

    no usages
    public void setProjectId(int projectId) { this.projectId = projectId; }

    no usages
    public int getSuccessfulBuilds() { return successfulBuilds; }

    no usages
    public void setSuccessfulBuilds(int successfulBuilds) { this.successfulBuilds = successfulBuilds; }

    no usages
    public int getFailedBuilds() { return failedBuilds; }

    no usages
    public void setFailedBuilds(int failedBuilds) { this.failedBuilds = failedBuilds; }

    no usages
    public double getAverageBuildTime() { return averageBuildTime; }

    no usages
    public void setAverageBuildTime(double averageBuildTime) { this.averageBuildTime = averageBuildTime; }
}
```

```java
public class CIServer {
    1 usage
    private List<Project> registeredProjects = new ArrayList<>();
    2 usages
    private ProjectRepository projectRepository;
    1 usage
    private BuildScriptRepository buildScriptRepository;
    2 usages
    private BuildStatisticsRepository buildStatisticsRepository;

    1 usage
    public CIServer() {
        // Ініціалізація репозиторіїв для проектів, build-скриптів та статистики
        projectRepository = new ProjectRepository();
        buildScriptRepository = new BuildScriptRepository();
        buildStatisticsRepository = new BuildStatisticsRepository();
    }

    1 usage
    public void registerProject(Project project) {
        // Реєстрація нового проекту на сервері та в базі даних
        registeredProjects.add(project);
        projectRepository.create(project);
    }

    no usages
    public void recognizeDependencies(Project project) {
        // Розпізнання залежностей між проектами
    }

    no usages
    public void compileProject(Project project) {
        // Компіляція проекту для обраної мови програмування
    }
```

```
    no usages
    public void runTests(Project project) {
        // Виконання тестування проекту
    }

    no usages
    public void recognizeBuildScript(Project project) {
        // Розпізнання та виконання build-скрипта
    }

    no usages
    public void updateBuildStatus(Project project, String status) {
        // Оновлення статусу збірки проекту
        project.setBuildStatus(status);
    }

    no usages
    public BuildStatistics generateStatistics(Project project) {
        // Генерація статистики для проекту та зберігання її в базу даних
        BuildStatistics statistics = new BuildStatistics(/* заповнити дані статистики */);
        buildStatisticsRepository.create(statistics);
        return statistics;
    }
}
```

```
3 usages   3 implementations
public interface Repository<T> {
    2 usages   3 implementations
    void create(T item);
    no usages   3 implementations
    void update(T item);
    no usages   3 implementations
    void delete(int itemId);
    no usages   3 implementations
    T getById(int itemId);
    no usages   3 implementations
    List<T> getAll();
}
```

```java
2 usages
public class ProjectRepository implements Repository<Project> {
    1 usage
    private List<Project> projects = new ArrayList<>();

    2 usages
    @Override
    public void create(Project project) {
        // Логіка для створення нового проекту в базі даних
    }

    no usages
    @Override
    public void update(Project project) {
        // Логіка для оновлення існуючого проекту в базі даних
    }

    no usages
    @Override
    public void delete(int projectId) {
        // Логіка для видалення проекту з бази даних за ідентифікатором
    }

    no usages
    @Override
    public Project getById(int projectId) {
        // Логіка для отримання проекту з бази даних за ідентифікатором
        return null;
    }

    no usages
    @Override
    public List<Project> getAll() {
        // Логіка для отримання всіх проектів з бази даних
        return projects;
    }
}
```

```java
2 usages
public class BuildScriptRepository  implements Repository<BuildScript> {
    1 usage
    private List<BuildScript> buildScripts = new ArrayList<>();

    2 usages
    @Override
    public void create(BuildScript buildScript) {

    }

    no usages
    @Override
    public void update(BuildScript buildScript) {

    }

    no usages
    @Override
    public void delete(int buildScriptId) {

    }

    no usages
    @Override
    public BuildScript getById(int buildScriptId) {

        return null;
    }

    no usages
    @Override
    public List<BuildScript> getAll() {

        return buildScripts;
    }
}
```

**Висновок:** виконуючи дану лабораторну робота я навчився створювати діаграму варіантів використання, сценарії варіантів використання, UML діаграми, діаграми класів, концемптуальну модель системи.