# Тестування та верифікація програмного забезпечення

## Лекція №6

## Тема: Реалізація програмного забезпечення

ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Лектор: асистент кафедри комп'ютерних наук Українець Микола Олександрович

# Питання лекції

1. Вимоги до оформлення коду: стиль, розбиття на структуровані одиниці, найменування змінних, класів, об'єктів
2. Налагодження
3. Засоби автоматичної генерації програмного коду
4. Керування конфігурацією та версіями програмного забезпечення
5. Постійна інтеграція/постійне впровадження (Continuous Integration/Continuous Delivery).

# Вимоги до оформлення коду

Стандарт оформлення коду (англ. coding standards, англ. coding convention або англ. programming style) — набір правил та угод, що використовуються при написанні програмного коду.

Метою прийняття та використання стандарту є **спрощення сприйняття програмного коду людиною, мінімізація навантаження на пам'ять та зір при читанні програми**.

# Вимоги до оформлення коду

Зазвичай, стандарт оформлення коду описує:

- засоби вибору значень та використовуваний регістр символів для імен змінних та інших ідентифікаторів.
- запис типу змінної в її ідентифікаторі (угорська нотація).
- регістр символів, використання знаку підкреслення для розділу слів.
- стиль відступів при оформленні логічних блоків — чи використовуються табуляції, ширина відступу.
- спосіб розстановки дужок, що обмежують логічні блоки.
- використання пробілів при оформленні логічних та арифметичних виразів.
- стиль коментарів та використання документуючих коментарів.

# Вимоги до оформлення коду

**Узгодження імен** — це набір правил для вибору послідовності символів, які будуть використовуватися для ідентифікаторів, які позначають змінні, типи, функції та інші сутності у початковому коді та документації.

Чим це корисно:

- Щоб зменшити зусилля, необхідні для читання та розуміння коду;
- Щоб перевірки коду зосередилися на питаннях, важливіших за синтаксис і стандарти імен..

# Вимоги до оформлення коду

Деякі з потенційних **переваг**, які можна отримати, прийнявши угоду про іменування, включають наступне:

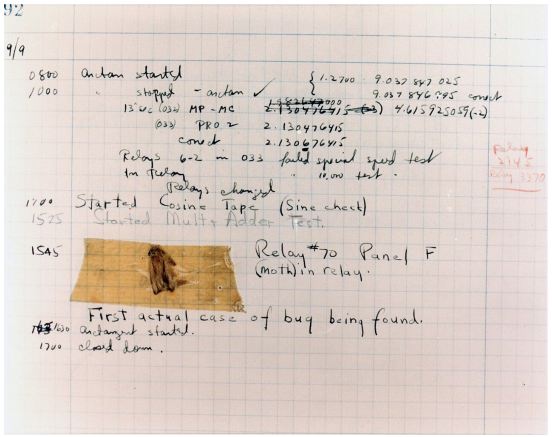
- надати додаткову інформацію (тобто метадані) про використання ідентифікатора;
- допомогти формалізувати очікування та сприяти несуперечності в команді розробників;
- уможливити використання автоматизованого рефакторингу або інструментів пошуку та заміни з мінімальною можливістю помилок;
- для підвищення ясності у випадках потенційної двозначності;
- покращити естетичний і професійний вигляд продукту роботи (наприклад, заборонивши надто довгі назви, смішні чи «милі» назви або абревіатури);
- щоб допомогти уникнути «колізій імен», які можуть виникнути, коли робочий продукт різних організацій об'єднується (див. також: простори імен);
- надавати значущі дані для використання під час передачі проекту, що вимагає подання вихідного коду програми та всієї відповідної документації;
- щоб забезпечити краще розуміння у випадку повторного використання коду після тривалого проміжку часу.

# Вимоги до оформлення коду

| Naming Convention | Example Format | Common Usage |
|---|---|---|
| **Pascal Case** | PascalCase | Classes (C#, Java, Python) |
| **Camel Case** | camelCase | Variables and Functions (JS, Java) |
| **Snake Case** | snake_case | Variables and Database Fields (Python, SQL) |
| **Kebab Case** | kebab-case | URLs and CSS Class Names |
| **Flat Case** | flatcase | Rare; sometimes Package Names |
| **Upper Flat Case** | UPPERFLATCASE | Environment Variables (less common) |
| **Pascal Snake Case** | Pascal_Snake_Case | Specific API responses or legacy code |
| **Camel Snake Case** | camel_Snake_Case | Rarely used; specific edge cases |
| **Screaming Snake Case** | SCREAMING_SNAKE_CASE | Global Constants |
| **Train Case** | Train-Case | HTTP Header Keys |
| **Cobol Case** | COBOL-CASE | Legacy systems / COBOL programming |

# Налагодження

**Налагодження програми (англ. debugging)** — методичний процес пошуку та зменшення числа помилок або дефектів у комп'ютерній програмі або електронному обладнанні з метою отримання очікуваної поведінки.




# Засоби автоматичної генерації програмного коду

**Автоматична генерація коду** — це використання інструментів для створення програмного коду на основі моделей, шаблонів, конфігурацій або метаданих.

- Скаффолдинг (Scaffolding) — автоматичне створення базової структури проєкту: контролерів, моделей та представлень (Views) на основі схеми бази даних (наприклад, Автоматичне створення контролерів, моделей, представлень, CRUD-операції за структурою БД, Генерація REST API за описом сутностей, контролери в ASP.NET, генерація ресурсів у веб-фреймворках).
- Генерація на основі контрактів — це підхід до розробки, за якого першочергово створюється формальний, незалежний від мови програмування опис інтерфейсу (контракт/специфікація), на основі якого автоматично генерується програмний код для клієнта та сервера.
- Низькорівневі препроцесори та кодогенератори — інструменти, що створюють допоміжний код «під капотом» для зменшення рутини (наприклад, Lombok у Java для геттерів/сеттерів або Protobuf для мережевих протоколів).

# Засоби автоматичної генерації програмного коду

- Model-Driven Development (MDD) — генерація програмного коду безпосередньо з візуальних діаграм (наприклад, UML-схем) або високорівневих бізнес-моделей.
- Low-Code / No-Code платформи — середовища для створення функціональних застосунків через графічний інтерфейс, де код генерується автоматично на основі логічних блоків.
- ШІ-асистенти (AI Coding Assistants) — інструменти на базі нейромереж (GitHub Copilot, Tabnine), що генерують логічні блоки або цілі функції на основі контексту та описів природною мовою.

# Налагодження

**Налагоджувач (Debugger)** — спеціалізоване програмне забезпечення для контролю за процесом виконання програми (покрокове виконання, зупинка, інспектування пам'яті).

**Точки зупинки (Breakpoints)** — навмисно встановлені маркери в коді, на яких виконання програми призупиняється, щоб розробник міг перевірити поточний стан системи.

**Спостереження за змінними (Variable Watch)** — функція налагоджувача, що дозволяє відстежувати зміну значень конкретних змінних у реальному часі під час виконання коду.

**Виведення на консоль (Console Output)** — базовий метод налагодження шляхом друку текстових повідомлень (логів) у термінал для відстеження логіки роботи програми.

**Аналізатори коду (Code Analyzers)** — інструменти для автоматичної перевірки синтаксису, стилю та пошуку потенційних вразливостей або помилок без фактичного запуску коду (статичний аналіз).

**Порада для лектора:** Налагоджувач — це ваш "рентген", консоль — це ваші "записки на полях", а аналізатори — це "коректор", який б'є по руках ще до того, як ви зробите помилку.

# Керування конфігурацією та версіями програмного забезпечення

**Конфігурація системи** — це склад функцій, програмного і технічного забезпечення системи, можливі їх комбінації залежно від наявності устаткування, загальносистемних засобів і вимог до продукту.

**Керування конфігурацією** — це ідентифікація компонентів системи, визначення функціональних, фізичних характеристик системи, апаратного і програмного забезпечення для контролю виконання, внесення змін і трасування конфігурації.

Роботи з конфігураційного керування програмного забезпечення включають:

- управління і планування SCM-процесів,
- ідентифікацію програмних конфігурацій,
- контроль конфігурацій,
- облік статусів конфігурацій,
- аудит, а також
- управління випуском (release management)
- і поставкою (delivery).

# Керування конфігурацією та версіями програмного забезпечення

**Система керування версіями** — це інструмент, який забезпечує збереження історії змін у файлах і координацію роботи розробників.

**Основні можливості:**

- Історія змін (хто, коли, що змінив)
- Повернення до попередніх версій
- Паралельна розробка (гілки)
- Злиття змін (merge)
- Розв’язання конфліктів


Типи систем контролю версій:

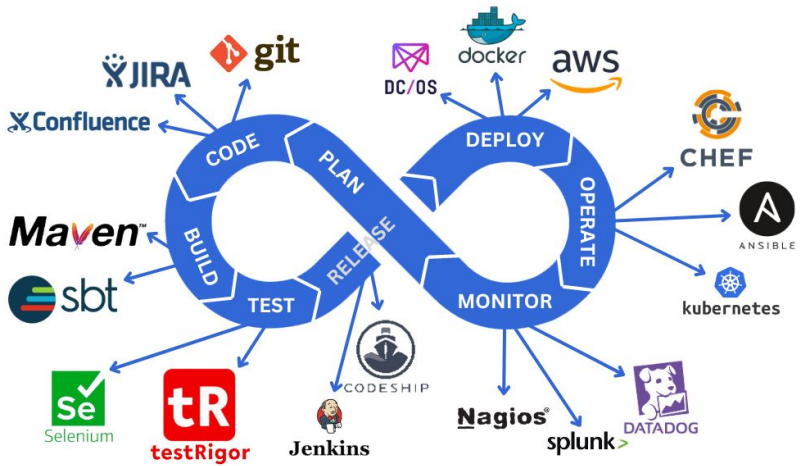
- Централізовані (CVCS) — один центральний репозиторій
- Розподілені (DVCS) — кожен розробник має повну копію репозиторію

# Безперервна інтеграція (CI) та безперервна доставка та розгортання (CD)

**CI/CD (Continuous Integration/Continuous Delivery або Continuous Deployment)** — це набір практик у розробці програмного забезпечення, що автоматизують процеси злиття коду, його тестування та розгортання для прискорення та підвищення надійності випуску оновлень.

**CI (безперервна інтеграція)** означає часте об'єднання коду від різних розробників у спільний репозиторій з автоматичним запуском тестів для виявлення помилок.

**CD (безперервна доставка або розгортання)** завершує процес, автоматично готуючи код до релізу (доставка) або відправляючи його на продакшн (розгортання), що забезпечує швидше та якісніше постачання продукту користувачам.



# Безперервна інтеграція (CI) та безперервна доставка та розгортання (CD)

**Основні принципи CI/CD:**

- Зниження ризиків у процесі.
- Організація середовища для всіх, хто в ньому працює.
- Швидкий зворотний зв’язок через автоматизацію процесів та мінімізацію посередників.
- Рівномірний розподіл відповідальності між працівниками.

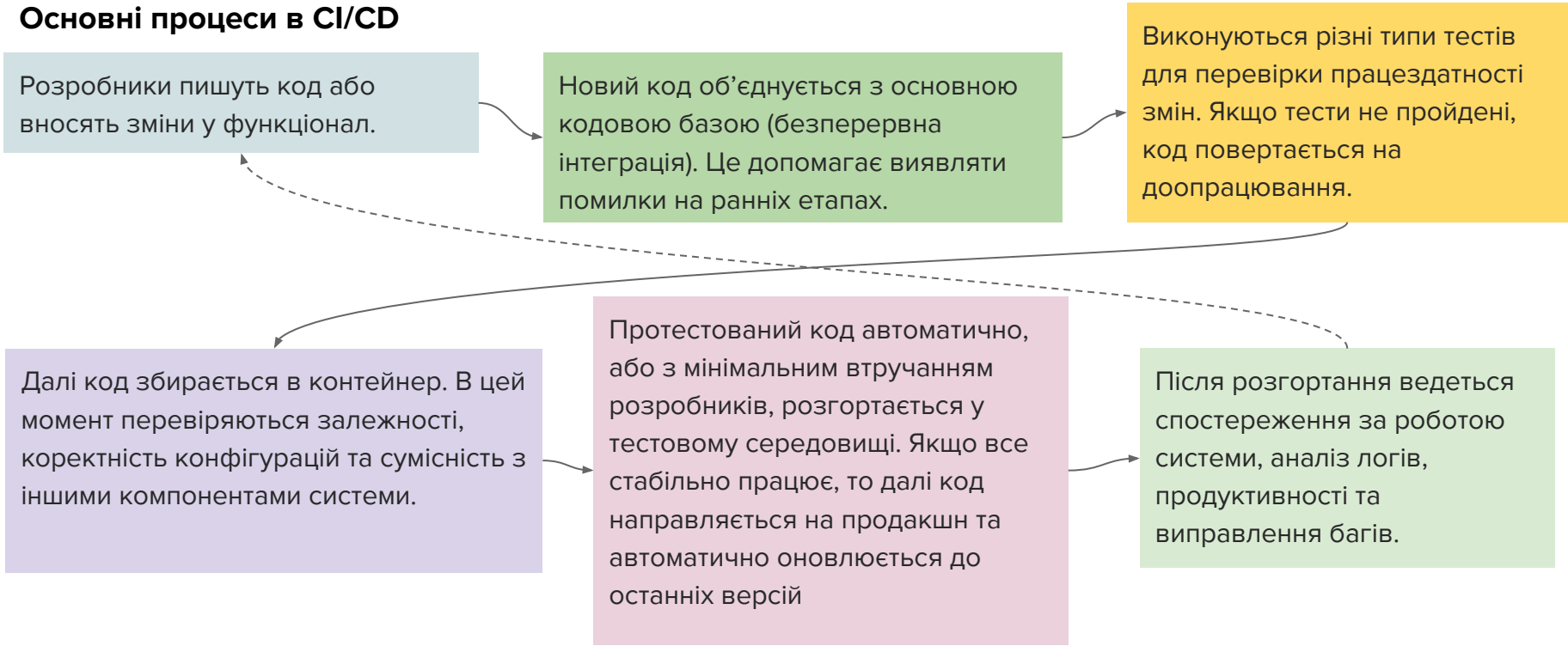
**CI** означає, що кожна зміна у коді має бути:
об’єднана з основною гілкою (merge);
автоматично зібрана (build);
протестована.

**Ціль — раннє виявлення дефектів і забезпечення стабільності коду.**

**CD** — це автоматизація процесу доставки програмного забезпечення:
Continuous Delivery — автоматично готує реліз, але вимагає ручного підтвердження для деплою;
Continuous Deployment — здійснює деплой повністю автоматично.

# Безперервна інтеграція (CI) та безперервна доставка та розгортання (CD)

**Основні процеси в CI/CD**



# Безперервна інтеграція (CI) та безперервна доставка та розгортання (CD)

**Конвеєр безперервної інтеграції та безперервного розгортання** (CI/CD pipeline) — це послідовність встановлених кроків, яких розробники повинні дотримуватися для випуску нової версії програмного забезпечення.

**CI/CD-пайплайни** — це практика, зосереджена на покращенні доставки програмного забезпечення протягом усього життєвого циклу його розробки за допомогою автоматизації.

У деталях конвеєр CI/CD може відрізнятися від проєкту до проєкту: залежно від програмного забезпечення, яке створюють розробники, самої команди розробників, наявних ресурсів тощо.

Типовий pipeline складається з кількох етапів:

| Етап | Завдання |
|---|---|
| **Checkout** | Отримання коду з репозиторію |
| **Build** | Компіляція, збірка |
| **Test** | Запуск юніт, інтеграційних і UI-тестів |
| **Analysis** | Аналіз коду, покриття, безпека |
| **Deploy** | Розгортання на staging або production |
| **Monitor** | Моніторинг, логування |

# Дякую за увагу!