УДК 004.5
DOI https://doi.org/10.32782/2521-6643-2025-2-70.24

Дьячук Т. С., старший викладач кафедри комп'ютерних систем та мереж Національного університету «Запорізька політехніка»
ORCID: 0000-0002-2478-0588

Скрупський С. Ю., кандидат технічних наук, доцент, доцент кафедри комп'ютерних систем та мереж Національного університету «Запорізька політехніка»
ORCID: 0000-0002-9437-9095

Голуб Т. В., кандидат технічних наук, доцент, доцент кафедри комп'ютерних систем та мереж Національного університету «Запорізька політехніка»
ORCID: 0000-0001-6024-008X

АВТОМАТИЗОВАНА СИСТЕМА ПЕРЕВІРКИ ЗАВДАНЬ ДЛЯ НАВЧАЛЬНИХ КУРСІВ ІЗ ПРОГРАМУВАННЯ

Метою статті є представлення результатів розробки та впровадження автоматизованої системи перевірки завдань, яка допоможе викладачу організувати якісний, об'єктивний та ефективний навчальний процес в умовах асинхронного дистанційного навчання, зумовленого спочатку пандемією COVID-19, а згодом повномасштабною війною. Ручна перевірка великої кількості програмних робіт створює значне навантаження на викладачів, призводить до затримок у наданні зворотного зв'язку та вносить елементи суб'єктивності в оцінювання. Розроблена система є компонентом ширшої освітньої платформи, в якій завдання для студентів вже генеруються індивідуально. Запропонована архітектура перевірки базується на інтеграції інструментів контролю версій та безперервної інтеграції. Процес роботи організовано наступним чином: студент виконує завдання у своєму репозиторії та створює Pull Request для злиття змін. Ця дія автоматично запускає процес перевірки (GitHub Action), який компілює код та передає його валідатору. Валідатор генерує великий набір тестових даних, виконує на них як код студента, так і еталонну реалізацію (описану формалізованими формулами), а потім порівнює результати. У разі розбіжностей система миттєво надає студенту звіт про помилку з конкретним набором даних, що її спричинив. Це дозволяє викладачу зосередитись не на рутинній перевірці коректності, а на аналізі якості коду, архітектури та наданні змістовних коментарів. Система реалізована мовою Kotlin, використовує систему збірки Gradle та поширюється у вигляді бібліотеки (Maven артефакту), що підключається студентами до своїх проєктів. Апробація проводилась у межах дисципліни «Основи програмування на Kotlin» за участю 23 студентів. Впровадження системи довело свою ефективність у заощадженні часу викладача, забезпеченні об'єктивного оцінювання та підвищенні мотивації студентів до експериментів та самостійного виправлення помилок завдяки миттєвому зворотному зв'язку. Подальший розвиток системи передбачає інтеграцію модулів для оцінки якості коду.

Ключові слова: *GitHub, Kotlin, Gradle, Maven артефакт, автоматизована перевірка, дистанційна освіта, репозиторій.*

Diachuk T. S., Skrupsky S. Yu., Holub T. V. Automated assignment grading system for programming courses

The purpose of the article is to present the results of the development and implementation of an automated assignment grading system designed to help instructors organize a high-quality, objective, and effective educational process in an asynchronous distance learning environment. The relevance of the work is driven by the challenges of distance education in Ukraine, caused first by the COVID-19 pandemic and later by the full-scale war, which demands the implementation of effective, flexible, and secure educational tools capable of functioning in a predominantly asynchronous learning environment. Manual grading of numerous programming assignments creates a significant burden on instructors, leads to delays in providing feedback, and introduces elements of subjectivity into the assessment. An analysis of recent research confirms a global trend toward assessment automation in IT education as a means of increasing efficiency and objectivity. The developed system is a component of a broader educational platform in which assignments for students are already generated individually. The proposed grading architecture is based on the integration of version control and continuous integration tools. The workflow is organized as follows: a student completes an assignment in their repository and creates a Pull Request to merge the changes. This action automatically triggers the verification process (a GitHub Action), which compiles the code and passes it to a validator. The validator generates a large

set of test data, executes both the student's code and a reference implementation (described by formalized formulas) on this data, and then compares the results. In case of discrepancies, the system instantly provides the student with an error report, including the specific data set that caused the failure. This allows the instructor to focus not on routine correctness checks, but on analyzing code quality, architecture, and providing meaningful comments. The system is implemented in Kotlin, uses the Gradle build system, and is distributed as a library (a Maven artifact) that students connect to their projects. Approbation was conducted within the «Basics of Programming in Kotlin» course with the participation of 23 students. The implementation of the system proved its effectiveness in saving instructor time, ensuring objective assessment, and increasing student motivation to experiment and independently correct errors, thanks to the instant feedback. Future development of the system includes the integration of modules for assessing code quality.

Key words: *GitHub, Kotlin, Maven artifact, Gradle, automated grading, distance education, repository.*

Постановка проблеми. Сучасний етап розвитку вищої освіти характеризується активним упровадженням інформаційно-комунікаційних технологій та цифрових освітніх сервісів. Це зумовлює необхідність пошуку інноваційних підходів до організації навчального процесу, серед яких особливої актуальності набуває автоматизована генерація та перевірка навчальних завдань як складова цифровізації освітнього середовища.

В останні роки дистанційна форма навчання набула значного поширення. Її активне використання було зумовлене пандемією COVID-19, а згодом – повномасштабною війною в Україні, що особливо актуалізувало потребу у впровадженні сучасних цифрових освітніх сервісів, зокрема у прифронтових регіонах. Крім того, виклики, спричинені регулярними відключеннями електроенергії та частими повітряними тривогами, зумовили необхідність переходу до переважно асинхронного формату навчання. Це дозволило учням та студентам отримувати доступ до навчальних матеріалів і виконувати завдання у зручний для них час, незалежно від графіків стабілізаційних відключень чи потреби перебувати в укритті. Водночас дистанційна освіта стає самостійним і перспективним освітнім форматом, який завдяки розвитку цифрових технологій забезпечує доступність, мобільність, гнучкість та безпеку освітнього процесу.

Одним із провідних напрямів цифровізації освіти є впровадження автоматизованих засобів підтримки навчальної діяльності, зокрема систем для генерації та перевірки завдань. У статті розглянуто автоматизовану систему перевірки завдань для навчальних курсів із програмування. Вона дозволяє ефективно контролювати виконання завдань студентами, автоматично перевіряти правильність коду, виявляти помилки та надавати зворотний зв'язок без необхідності ручної перевірки кожного завдання. Такий підхід дає змогу заощаджувати час викладачів та студентів, сприяє швидшому виявленню та виправленню помилок, а також забезпечує об'єктивніше оцінювання робіт. Це потужний інструмент для оптимізації освітнього процесу, підвищення ефективності викладання та полегшення навчання, який забезпечує перевірку правильності коду, якості реалізації та відповідності вимогам без постійної участі викладача.

Запропонована система надає ряд переваг, особливо в умовах дистанційної освіти. Так, автоматизація допомагає впоратися з первинною перевіркою великої кількості типових завдань та заощадити час як викладача, так і студента. Студенти можуть отримати результати перевірки та зворотний зв'язок одразу після здачі роботи й швидше виправити свої помилки, що мотивує до самостійного навчання, а також спонукає виконувати завдання згідно з чіткими вимогами. Такий підхід є доречним для великих груп студентів, де викладачам важко вручну перевірити кожне завдання. Це зменшує навантаження на викладачів і дозволяє їм більше зосередитися на індивідуальній допомозі студентам. Також виключається фактор людської суб'єктивності в оцінюванні, оскільки автоматизована система забезпечує стандартизовану перевірку.

Однак такий підхід має і певні недоліки. Розробка та підтримка подібної системи потребує значних витрат часу, а також кваліфікації для налаштування тестів і критеріїв оцінювання. Система може перевіряти лише чітко задані показники (наприклад, правильність вихідних даних), але не здатна оцінити творчі або нестандартні підходи до розв'язання задачі. Для завдань, які вимагають креативних рішень, автоматизована система може бути використана лише для початкового аналізу коректності роботи. Крім того, якщо система має помилки або неправильно налаштовані параметри, це може призвести до некоректного оцінювання студентських робіт.

Аналіз останніх досліджень та публікацій. Актуальність та перспективність використання дистанційних платформ для навчання програмуванню підтверджується численними останніми дослідженнями у цій галузі. Наприклад, стаття [1] представляє «GRAD-AI» – вдосконалений інструмент для автоматичного оцінювання завдань з програмування, який поєднує можливості штучного інтелекту (ШІ) з контролем викладача для подолання недоліків ручної перевірки. Система використовує комплексні метрики (зокрема, складність Холстеда, TF-IDF та аналіз синтаксичних дерев) для надання миттєвого, точного та неупередженого зворотного зв'язку студентам. Автори роблять висновок, що «GRAD-AI» є значним кроком у вдосконаленні освітнього процесу, демонструючи потенціал ШІ у персоналізації навчання та виявленні прогалин у знаннях.

У дослідженні [2] неформальні лабораторні роботи з програмування замінили на щотижневі автоматизовані тести в системі HackerRank для студентів, які вивчали структури даних та алгоритми. Хоча студенти

негативно ставилися до нової системи, їхня успішність у HackerRank виявилася найкращим показником майбутніх результатів на іспиті. Впровадження цих тестів призвело до того, що загальний рівень незадовільних оцінок знизився вдвічі, а кількість відмінників – потроїлася. Автори роблять висновок, що автоматичне оцінювання краще готує студентів до самостійного написання коду, оскільки усуває залежність від сторонньої допомоги та мотивує до розвитку самодостатності.

Стаття [3] досліджує, як великі мовні моделі (LLM) можуть автоматизувати оцінювання завдань у вступному курсі з програмування, вирішуючи проблеми часових затрат та непослідовності ручної перевірки. Дослідження дійшло висновку, що LLM під наглядом людини здатні значно скоротити робоче навантаження (з 300+ годин до 15 хвилин) та підвищити ефективність оцінювання в комп'ютерній освіті.

Отже, проведений аналіз наукових праць і сучасних розробок показує, що нині існує широкий спектр систем, методик та технологічних рішень, спрямованих на автоматизацію процесів навчання, зокрема у галузі програмування. Значна увага дослідників приділяється питанням підвищення ефективності навчання, оптимізації оцінювання та забезпечення інтерактивної взаємодії між студентом і навчальним середовищем.

Разом із тим, питання створення та впровадження автоматизованих систем перевірки завдань для навчальних курсів із програмування залишається надзвичайно актуальним. Такі системи забезпечують об'єктивність оцінювання, скорочують час перевірки результатів, а також сприяють індивідуалізації навчального процесу. Отже, тема розроблення й удосконалення автоматизованих засобів контролю знань у курсах програмування є затребуваною, перспективною та потребує подальших наукових досліджень.

Метою статті є розробка та апробація автоматизованої системи перевірки завдань при навчанні програмуванню, яка допоможе викладачу організувати ефективний та якісний навчальний процес в умовах дистанційної освіти, часто асинхронної через воєнний стан. Розроблювана система є частиною більш складної системи для дистанційної освіти, тому завдання, що перевіряються, спочатку генеруються автоматизованою системою генерації завдань, відповідають певним критеріям та мають чітко визначену специфікацію [4].

Виклад основного матеріалу. На (рис. 1) наведена архітектура автоматизованої системи перевірки завдань, яка була розроблена та апробована у рамках дисципліни «Основи програмування на Kotlin» (базовий репозиторій для навчання доступний за посиланням [5]). Система побудована на можливостях GitHub, таких як Pull requests (PR) та GitHub Actions. Git та GitHub є дуже потужними інструментами для контролю версій та спільної роботи над проєктами. Автоматизована система перевірки є частиною більш складної системи для дистанційної освіти, тому завдання, які надаються студентам, також генеруються автоматично за допомогою хеш-функції від ідентифікатора студента, що забезпечує унікальність варіантів [4]. Завдання придатні для подальшої автоматизованої перевірки та мають чітко визначену специфікацію.

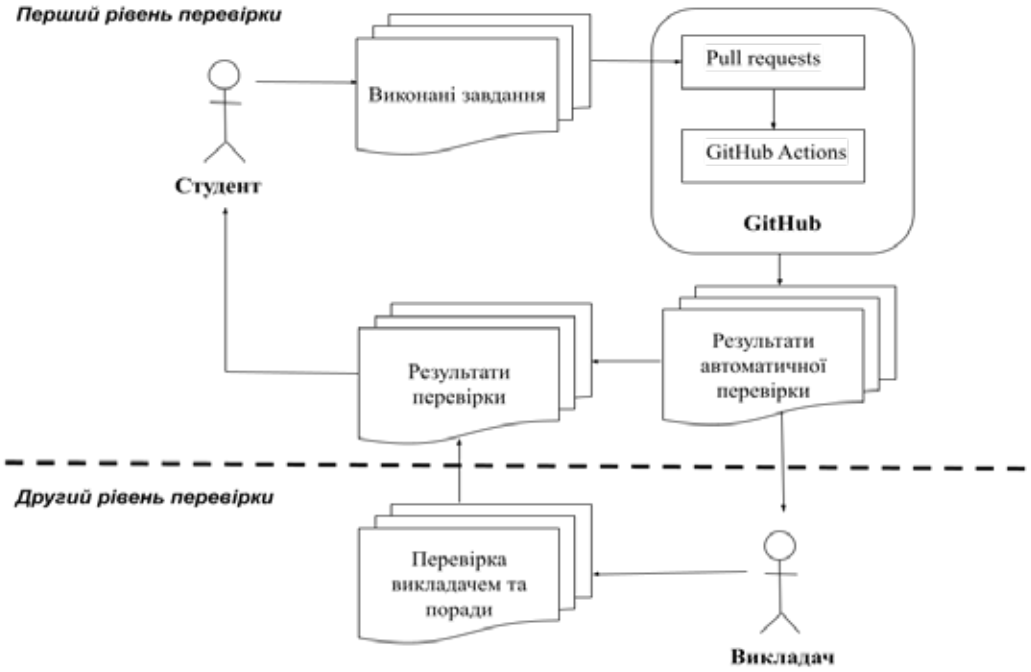

Рис. 1. Архітектура автоматизованої системи перевірки завдань для навчальних курсів із програмування

Принцип роботи автоматизованої системи перевірки полягає в наступному. Після виконання завдання для кожної нової функції або задачі, студенти створюють PR, щоб викладач міг переглянути та залишити коментар. GitHub Actions дозволяє використовувати засоби автоматичного тестування після змін у коді, що забезпечує моментальну відповідь студенту. При автоматичному тестуванні валідатор генерує набори вхідних даних, наприклад 100 значень, проходить цей набір і кожне значення віддає на блок виконання завдання та програмний код студента, потім порівнює результати. Якщо вони не співпадають, то вважається, що студент припустив помилку. Але завжди є можливість студенту звернутись до викладача. Викладач зі свого боку має результати автоматичної перевірки, не витрачає зайвого часу на рутинну роботу та може переглянути код та залишити свої зауваження, додаткові коментарі та поради з удосконалення коду.

Таким чином, у системі реалізовано два рівні взаємодії у процесі перевірки завдань. Перший рівень – автоматизована перевірка (GitHub Actions). Вона забезпечує швидке тестування коду, оперативний зворотний зв'язок і виявлення типових помилок без участі викладача. Другий рівень – перевірка викладачем. На цьому етапі відбувається глибший аналіз роботи студента, надання рекомендацій, коментарів та порад щодо покращення коду і розуміння логіки програмування. Поєднання автоматичного та ручного підходів забезпечує ефективність процесу перевірки, дозволяючи автоматизувати рутинні завдання та водночас зберегти індивідуальний підхід до кожного студента.

Для прикладу розглянемо перевірку завдання з теми «Змінні та типи даних, рядки, умови та цикли, функції». Для об'єднання та формалізації варіантів формул були використані елементи принципу One-Hot Encoding [6]. Оскільки при формуванні завдання *res* (формула (1)) використовується головна *mainFnc* та вторинна функції *secondaryFnc* [4], то для перевірки також потрібно враховувати всі наявні варіанти.

$$res = mainFnc\left(secondaryFnc\left(X_0, X_1, \ldots, X_n\right)\right), \tag{1}$$

де $\overline{X}$ – вектор вхідних аргументів розмірністю *n*, які для тестування система перевірки підбирає за допомогою генератора випадкових чисел.

Наведемо формулу (2) для розрахунку *secRes* – вектора результатів вторинної функції для відповідних вхідних аргументів *Xi*.

$$\begin{aligned} secRes_0 &= A_0 \cdot X_0 + A_1 \cdot X_0^2 + A_2 \cdot X_0^3 + A_3 \cdot X_0 + A_4 \cdot X_0 + A_5 \cdot |X_0| + \\ &+ A_6 \cdot |X_0| + A_7 \cdot |X_0| \\ secRes_i &= secRes_{i-1} \cdot sum + secRes_{i-1}^{mult} \cdot (A_0 \cdot X_i + A_1 \cdot X_i^2 + A_2 \cdot X_i^3 + \\ &+ A_3 \cdot \min(secRes_{i-1}, X_i) + A_4 \cdot \max(secRes_{i-1}, X_i) + A_5 \cdot |X_i| + \\ &+ A_6 \cdot \min(secRes_{i-1}, |X_i|) + A_7 \cdot \max(secRes_{i-1}, |X_i|)), \end{aligned} \tag{2}$$

де *A* – вектор, в якому є лише один ненульовий елемент, який буде у позиції відповідно до завдання студента. Наприклад, якщо є завдання розрахувати суму квадратів, то вектор буде наступним *A = [0, 1, 0, 0, 0, 0, 0]*;

sum – буде мати значення *1*, якщо в завданні потрібно розрахувати суму всіх компонентів, інакше 0;

mult – буде мати значення *1*, якщо в завданні потрібно перемножити всі компоненти, інакше 0;

i – приймає значення від *1* до *n*.

Головна функція *result* розраховується аналогічно (3).

$$\begin{aligned} data = [&\sqrt{secRes_n}, \sqrt[3]{secRes_n}, \cos\left(secRes_n\right), \tan\left(secRes_n\right), \\ &\sin\left(secRes_n\right), \tanh\left(secRes_n\right), \ln\left(secRes_n\right)] \\ result = &\overrightarrow{data} \cdot \vec{B}, \end{aligned} \tag{3}$$

де *B* – вектор, в якому є лише один ненульовий елемент, він буде у позиції відповідно до головного завдання студента. Наприклад, якщо студент отримав завдання розрахувати корінь квадратний з суми квадратів, то вектор буде наступним *B = [1,0,0,0,0,0,0]*;

$secRes_n$ – останній елемент вектору *secRes*, тобто результат ітеративного розрахунку значення вторинної функції завдання;

data – вектор всіх можливих головних функцій завдання.

На всі завдання система генерує набори тестових даних. Далі виконується розрахунок результату за кожним елементом тестових даних за програмним кодом від студента та з використанням вищенаведених формул. Якщо результати не співпадають, то завдання студента не зараховується, система видає звіт про помилку з певним набором тестових даних (приклад представлено в лістингу 1).

Лістинг 1. Звіт про результати перевірки завдання (автоматичне тестування):

```
Lab2Test > task2Test() FAILED
java.lang.AssertionError: Не співпадають результати набору
"x0 = 3.9579079629338776, x1 = 3.4867284435369874,
x2 = -2.026071933406628, x3 = -6.871367991204648, ",
перевірте завдання 2),
додайте функцію dCalculate() : Double,
яка отримує наступні аргументи:
   - x0 типу Double зі значенням за замовчуванням -52.48
   - x1 типу Double зі значенням за замовчуванням 15.12
   - x2 типу Double зі значенням за замовчуванням 0.66
   - x3 типу Double зі значенням за замовчуванням 0.84
. Expected <13.86088440642907>, actual <12.34>.
```

Такий механізм дозволяє забезпечити об'єктивність і прозорість оцінювання, оскільки всі перевірки виконуються за єдиним алгоритмом без участі викладача. У процесі автоматичного тестування система може виявляти помилки, що виникають через невідповідність отриманих результатів очікуваним. Наприклад, під час перевірки функції dCalculate() виникла помилка типу AssertionError, яка вказує, що результати обчислень не збігаються із заданими еталонними значеннями (лістинг 1). Це свідчить про те, що реалізація функції містить неточність у формулі або у використанні вхідних аргументів. Зокрема, тест очікував значення 13.86088440642907, тоді як фактичний результат становив 12.34. Подібні звіти дозволяють швидко визначити джерело помилки, що суттєво скорочує час на налагодження коду та підвищує ефективність навчального процесу. Студенти отримують детальний опис проблеми та можуть самостійно виправити свій код, а викладач позбавляється необхідності вручну перевіряти кожне завдання, зосереджуючись натомість на аналізі типових помилок і наданні рекомендацій.

Для реалізації системи було використано такі інструменти: мову програмування Kotlin [7] для розробки, систему автоматизації збірки Gradle, середовище розробки IntelliJ IDEA [8], а також систему керування версіями Git і репозиторій GitHub [5] для зберігання вихідного коду. Скомпільована версія системи розміщена в репозиторії GitHub і доступна студентам у вигляді бібліотеки, яку можна підключати до власних проєктів під час вивчення навчального матеріалу. Використання Git і GitHub забезпечує ефективний контроль версій, спрощує командну роботу та сприяє організованому процесу розробки. Бібліотека зроблена у вигляді Maven артефакту [5] – стандартизованого формату для поширення скомпільованого коду в екосистемі Java/Kotlin. Такий підхід спрощує процес інтеграції для студентів, оскільки вони підключають бібліотеку до своїх проєктів через систему збірки Gradle, а не вручну. Це також дозволяє авторам системи легко випускати оновлення та виправлення шляхом зміни версії артефакту. Для підключення бібліотеки в навчальних проєктах потрібно у конфігураційний файл власного проєкту (build.gradle) додати шлях до репозиторію з Maven артефактами та увімкнути залежність від com.diacht.ktest:library бібліотеки.

Щоб оцінити ефективність автоматизованої системи перевірки завдань, проведено дослідження із групою з 23 студентів, які вивчали дисципліну «Основи програмування на Kotlin». Кожен студент отримував завдання в межах п'яти різних робіт, що відрізнялися за рівнем складності та тематичною наповненістю.

У таблиці 1 подано результати тестування, які відображають показники ефективності для кожного завдання.

Таблиця 1

Статистика перевірки завдань

№	назва	складність	бали	число спроб	число вирішень	% вирішень від спроб	% вирішень серед студентів
1	2.1	1	8	27	21	77,78	91,30
2	2.2	1	8	28	21	75,00	91,30
3	2.3	3	9	40	16	40,00	69,57
4	3.1	6	27	8	2	25,00	8,70
5	4.1	4	40	39	18	46,15	78,26

Аналіз результатів виконання завдань із програмування за допомогою автоматизованої системи перевірки показав її високу ефективність у навчальному процесі. Прості завдання (2.1–2.2) мали найвищий рівень успішності – понад 90 % студентів виконали їх правильно, що свідчить про доступність початкового рівня. Завдання середньої складності (2.3) викликали найбільшу активність (40 спроб), що вказує на оптимальний баланс між складністю та досяжністю. Найскладніше завдання (3.1) було успішно виконане лише

8,7 % студентів, що демонструє потребу у вдосконаленні механізмів адаптації складності та наданні додаткової підтримки під час розв'язання таких задач. Висока кількість спроб свідчить про підвищену мотивацію студентів до повторного виконання, експериментування та самостійного вдосконалення рішень. Автоматизована перевірка дала змогу студентам отримувати миттєвий зворотний зв'язок без залучення викладача, що істотно зекономило час викладача і зняло навантаження, пов'язане з ручною перевіркою численних спроб.

Отже, система забезпечує об'єктивне оцінювання, підтримує індивідуалізований підхід до навчання, сприяє підвищенню мотивації студентів і водночас оптимізує роботу викладачів. У подальшому доцільно інтегрувати адаптивні алгоритми навчання для покращення результативності виконання завдань підвищеної складності.

Висновки та перспективи. Таким чином, впровадження автоматизованих систем для генерації та перевірки завдань у курсах з програмування дозволяє суттєво оптимізувати освітній процес, спрощуючи як методику викладання, так і процедуру оцінювання академічної успішності студентів. Системи автоматизованого тестування є валідним інструментом підвищення об'єктивності та ефективності контролю знань у процесі підготовки ІТ-фахівців. Їх імплементація дозволяє автоматизувати перевірку великого масиву типових завдань, що оптимізує часові ресурси викладача. Для студентів практичне застосування таких систем сприяє засвоєнню сучасних технологій розробки та набуттю навичок колективної роботи над програмними проєктами, що є ключовою передумовою їхньої майбутньої конкурентоспроможності. Перспективним напрямом роботи є розширення функціональності системи шляхом інтеграції модулів для оцінки якості коду за формалізованими критеріями (стандарти коду, якість документації тощо).

Автоматизована система перевірки завдань була розроблена, реалізована та апробована в межах навчальної дисципліни «Основи програмування на Kotlin». Вона охоплює різні теми навчання та передбачає генерацію індивідуальних завдань для кожного студента на основі його унікального ідентифікатора. Розроблена система є частиною більш складної системи для дистанційної освіти, яка попередньо генерує завдання з чітко визначеною специфікацією, що робить їх придатними для подальшої автоматизованої перевірки [4]. Система в скомпільованому виді викладена у GitHub репозиторій та доступна студентам у формі окремої бібліотеки, що приєднується до системи збірки Gradle [5]. Впровадження системи довело свою ефективність у заощадженні часу викладача, забезпеченні об'єктивного оцінювання та підвищенні мотивації студентів. Система постійно вдосконалюється завдяки регулярним оновленням репозиторію та усуненню виявлених недоліків. Подальший розвиток системи передбачає інтеграцію модулів для оцінки якості коду. Також планується додати можливість перевірки коду штучним інтелектом GitHub Copilot.

Сфера застосування системи охоплює освітні установи, зокрема університети, а також компанії, що організовують корпоративне навчання з програмування. Розроблена система може бути використана для підготовки ІТ-фахівців різних рівнів та інтегрована у сучасні освітні платформи.

Список використаних джерел:

1. Gambo I., Abegunde F. J., Gambo O. et al. GRAD-AI: An automated grading tool for code assessment and feedback in programming course. Educ Inf Technol 30, 9859–9899 (2025). https://doi.org/10.1007/s10639-024-13218-5 (дата звернення: 23.10.2025).

2. Maguire Ph., Maguire R., Kelly R. Using automatic machine assessment to teach computer programming, *Computer Science Education*, 2017, Vol. 27, Issue 3-4, P. 197–214, URL: https://doi.org/10.1080/08993408.2018.1435113 (дата звернення: 23.10.2025).

3. Cisneros-González, J., Gordo-Herrera, N., Barcia-Santos, I. and Sánchez-Soriano, J., 2025. JorGPT: Instructor-aided grading of programming assignments with large language models (LLMs). *Future Internet*, 17(6), p. 265. https://doi.org/10.3390/fi17060265 (дата звернення: 23.10.2025).

4. Дьячук Т.С. Автоматизована система генерації завдань в навчальних курсах з програмування / Т.С. Дьячук, В.І Шкрябець, А.В. Тіменко, Т.В. Голуб. *Вчені записки Таврійського національного університету імені В.І. Вернадського*. Серія: Технічні науки, Видавничий дім «Гельветика», Том 35 (74) № 2 2024. С. 85–90, https://doi.org/10.32782/2663-5941/2024.2/12 (дата звернення: 23.10.2025).

5. DiachT/KotlinLabsNUZP – Головний репозиторій з кодом. URL: https://github.com/DiachT/KotlinLabsNUZP (дата звернення: 23.10.2025).

6. Data Science in 5 Minutes: What is One Hot Encoding? URL: https://www.educative.io/blog/one-hot-encoding (дата звернення: 23.10.2025).

7. Pierre-Yves Saumont. The Joy of Kotlin: Manning Publications, 2019 – 480p.

8. IntelliJ IDEA. URL: https://www.jetbrains.com/idea (дата звернення: 23.10.2025).

References:

1. Gambo, I., Abegunde, F. J., Gambo, O. et al. (2025). GRAD-AI: An automated grading tool for code assessment and feedback in programming course. Educ Inf Technol 30, 9859–9899. https://doi.org/10.1007/s10639-024-13218-5 (date of access: 23.10.2025).

2. Maguire, Ph., Maguire, R., Kelly, R. Using automatic machine assessment to teach computer programming, *Computer Science Education*, 2017, Vol. 27, Issue 3-4, P. 197–214, https://doi.org/10.1080/08993408.2018.1435113 (date of access: 23.10.2025).

3. Cisneros-González, J., Gordo-Herrera, N., Barcia-Santos, I. and Sánchez-Soriano, J., (2025). JorGPT: Instructor-aided grading of programming assignments with large language models (LLMs). *Future Internet*, 17(6), p. 265. https://doi.org/10.3390/fi17060265 (date of access: 23.10.2025).

4. Diachuk, T. S. (2024). Automated system of tasks generation for the programming courses / Diachuk T.S., Shkriabets V.I., Timenko A.V, Holub T.V. *Scientific notes of Taurida National V.I. Vernadsky University*, Series: Technical Sciences, Publishing House «Helvetica», Vol. 35 (74) № 2, P. 85–90, https://doi.org/10.32782/2663-5941/2024.2/12 (date of access: 23.10.2025).

5. DiachT/KotlinLabsNUZP – Upstream repository. Retrieved from: https://github.com/DiachT/KotlinLabsNUZP (date of access: 23.10.2025).

6. Data Science in 5 Minutes: What is One Hot Encoding? Retrieved from: https://www.educative.io/blog/one-hot-encoding (date of access: 23.10.2025).

7. Pierre-Yves Saumont. The Joy of Kotlin: Manning Publications, 2019. 480p.

8. IntelliJ IDEA. Retrieved from: https://www.jetbrains.com/idea (date of access: 23.10.2025).

Дата надходження статті: 27.10.2025
Дата прийняття статті: 17.11.2025
Опубліковано: 30.12.2025