УДК 004.4'24
DOI https://doi.org/10.32782/2663-5941/2024.1.1/19

Дьячук Т.С.
Національний університет «Запорізька політехніка»

Шкрябець В.І.
Національний університет «Запорізька політехніка»

АВТОМАТИЗОВАНА СИСТЕМА ГЕНЕРАЦІЇ КОДУ НА МОВАХ ВИСОКОГО РІВНЯ

Стаття присвячена рішенню практичної задачі спрощення розробки застосунків, які мають працювати на багатьох платформах, за допомогою використання автоматизованої системи генерації коду на мовах високого рівня. Галузь використання системи – компанії з розробки програмного забезпечення. Розроблювана система є частиною більш складної системи машинного перекладу з одної мови високого рівня на іншу. Для досягнення поставленої мети було: проведено аналіз підходів оптимізації мультиплатформеної розробки, спроєктовано, розроблено та реалізовано автоматизовану систему генерації коду, впроваджено її у виробничий процес компанії з розробки програмного забезпечення, проведено аналіз отриманих результатів. Об'єктом дослідження є синтаксис мов програмування високого рівня. Предметом дослідження – автоматизована система генерації коду. Методи дослідження базуються на синтаксисі мов програмування високого рівня. Для реалізації системи використані: мова Kotlin для опису та розробки системи, система автоматизації збірки Gradle, XML формат для передачі проміжних результатів між різними етапами виконання генерації, середа розробки InteliJ Idea, система керування версіями файлів Git, GitHub – репозиторій для вихідного коду. Запропонована архітектура системи кодогенерації для подальшого масштабування має розділення на різні етапи обробки вхідних описів. Проміжні результати між етапами зберігаються у вигляді дерев, таких як абстрактне синтаксичне дерево, дерево розбору, стилістичне дерево. У результаті роботи користувач отримує вихідні файли згенерованого коду. В процесі проектування системи була розроблена метамова, яка містить засоби для декларативного опису структур, інтерфейсів, типів даних та методів. Запропонована система генерації коду з опису на метамові може генерувати структури та інтерфейси (примітиви) на мовах Kotlin, C/C++, Java, Swift з мінімальним втручанням людини. Кодогенератор викладено у вільний доступ для ознайомлення на платформі GitHub. Будь хто може використовувати розроблену систему у своїх проєктах по ліцензії GPL – 3.0 та запропонувати свої правки та покращення за допомогою механізмів пул-реквестів або створення запитів про помилки на GitHub.

Ключові слова: *github, kotlin, метамова, абстрактне синтаксичне дерево, дерево розбору, кодогенерація, ОС, стилістичне дерево.*

Постановка проблеми. Розробка мультиплатформеного застосунку, тобто під різні операційні системи (ОС), зазвичай означає, що одну і ту ж саму логіку потрібно буде писати декілька разів для кожної з обраних ОС на притаманній мові програмування. Це призводить до зайвого витрачання людських та фінансових ресурсів. Великі проєкти можуть використовувати декілька мов програмування в різних частинах системи. Автоматизована система генерації коду може полегшити спільну розробку і підтримку таких проєктів, зменшити рутинну роботу для програмістів. Застосування кодогенерації дозволяє об'єднати функціонал різних мов програмування в одному проєкті, що може бути корисним для певних завдань або команд розробників. Таким чином, обрана тема є актуальною.

Аналіз підходів оптимізації мультиплатформеної розробки. Існує декілька підходів для пришвидшення мультиплатформенної розробки. Наприклад, написання спільної бізнес-логіки використовуючи загальнодоступну мову (як-то C/C++ [1]) або скрипти (Lua [2]). Іншим підходом є використання мультиплатформених фреймворків таких як: фреймворк Flutter [3], технологія Kotlin Multiplatform [4], фреймворк React Native [5], фреймворк Xamarin [6].

Аналіз показав, що у кожного з методів є свої прихильники та області використання, але ж не існує ідеального та універсального рішення. Взагалі це працює допоки застосунок нескладний та й має свої недоліки. Особливо неможливість, або складність використання таких підходів в умовах коли проєкт вже в розробці і треба частину

перевести на загальні рейки. Доволі часто це призводить до збільшення часу розробки, або ж може вийти непритаманний інтерфейс користувача. Також крім спеціалістів з розробки під конкретну ОС потрібні ще додаткові розробники зі спеціалізованим досвідом.

Також існують системи генерації коду [7], які допомагають автоматизувати процес розробки застосунків. Зазвичай потрібно описати необхідний код за певними правилами або за допомогою графічних інструментів, або за допомогою деякого абстрактного опису. Кожна з систем генерації коду може використовуватися у певній області та має свої обмеження та недоліки, такі як можливість генерації неефективного або неоптимального коду, потребу в ретельному тестуванні та валідації, а також складність налагодження та розуміння коду, згенерованого автоматично. Крім того, неправильна конфігурація генератора може призвести до неправильного або небезпечного коду.

Метою статті є спрощення мультиплатформеної розробки, за допомогою використання автоматизованої системи генерації коду на мовах високого рівня. Нас цікавить можливість генерації вихідного опису на кілька мов високого рівня для автоматизації та спрощення написання застосунків для різних платформ. Розроблювана автоматизована система генерації коду є частиною більш складної системи транскомпіляції [8], тобто системи машинного перекладу з одної мови високого рівня на іншу. Яка дасть спеціалістам такий інструмент, що дозволить працювати в комфортній для них середі-мові та використовувати код один одного.

Виклад основного матеріалу. Розглянемо етапи, з яких складається наша система генерації коду (рис. 1).

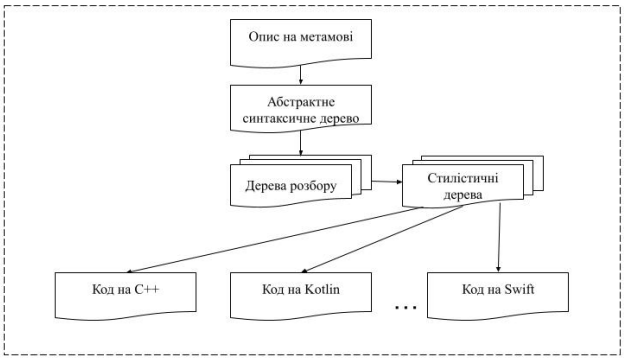

Рис. 1. Етапи генерації коду на мовах високого рівня

Спочатку необхідно зробити опис потрібного коду на метамові, який складається з одного чи декілька файлів. Метамова – це мова для опису загальної структури програми (логіки), яка базується на Kotlin Script. Вона була розроблена в процесі проєктування системи та являється проміжним етапом для відладки. Метамова містить засоби для декларативного опису структур, класів, інтерфейсів, типів даних, методів, тощо та дозволяє програмісту вивчати і змінювати опис структур в найбільш зручний для людини спосіб. Приклад декларування нескладної структури (клас даних в термінах Kotlin) GoldBuffer наведено у лістингу 1. Ця структура містить лише три поля даних: blockCount – ціле знакове число розміром 32 біта; blockSize – ціле знакове число розміром 32 біта; lastBlockSize – ціле знакове число розміром 32 біта, зі значенням за замовчуванням 10.

Лістинг 1. Декларування на метамові

```
1. namespace("com.goldman.data").apply {
2.  dataClass("GoldBuffer").apply {
3.  field("blockCount", DataType.int32)
4.  field("lastBlockSize", DataType.int32, 10)
5.  field("blockSize", DataType.int32)
6.  }
7. }
```

За описом на метамові формується абстрактне синтаксичне дерево (Abstract Syntax Tree або AST), наведено на рис. 2. AST є структурою даних, яка представляє синтаксичну структуру програми, відображаючи її синтаксичний аналіз у вигляді дерева. Тобто AST є інтермедіативним представленням програмного коду, яке дозволяє відокремити синтаксичну структуру від конкретних деталей синтаксису. Воно має наступну ієрархію рівнів: зверху простір імен; потім рівень класів, інтерфейсів; потім рівень методів та полів; рівень аргументів та логіки.

Кожен вузол в AST представляє конструкцію мови програмування (таку як оператор, вираз, об'єкт чи інше) і має дочірні вузли, які представляють його складові частини або аргументи. Дерево може бути використано для виконання різних видів аналізу програми. AST важливе для компіляторів та інших інструментів аналізу програмного коду. Воно допомагає зрозуміти структуру коду на рівні високого рівня, спрощує виконання оптимізації, перетворень та інших операцій над програмним кодом.

Як можна побачити дерево будується від кореневого вузла Namespace, яке починає декларувати простір імен. Від нього по ланцюжку ідуть простори імен “com”, “goldman”, “data”. Така організація дає можливість групувати класи, інтерфейси чи структури з одного простору імен в якості дочірніх елементів вузла. У нашому прикладі маємо структуру GoldBuffer, яка задекларована в просторі імен “com.goldman.data”. Вузол GoldBuffer

має три дочірніх вузла: поле даних blockSize, поле даних blockCount, поле даних lastBlockSize, як і було задекларовано на рівні метамови. Таким чином дерево повністю відповідає тому, що було описано на рівні метамови в лістингу 1.

Абстрактне синтаксичне дерево не можна використати для безпосередньої генерації вихідного коду оскільки воно ще занадто загальне та «віддалене» від результуючої мови. Приклад того, що може бути різним:

– організація файлів. В C/C++, прийнято декларування робити в .h файлах, а реалізацію в .c/.cpp. В мовах java/kotlin/swift/go декларування та реалізація в одному файлі;

– різна структура файлів. В Java/Kotlin простору імен декларується на початку файлу, потім йде блок імпорту. В мові C++ навпаки, в одному файлі може бути задекларовано багато просторів імен, а імпорт взагалі може бути в будь якому місці;

– структура директорій. В Kotlin/Java шлях до файлу на диску має співпадати з простором імен. В інших мовах таке не потрібно.

Тому потрібен наступний етап, на якому абстрактне дерево буде перетворено в дерево розбору (рис. 3). Дерево розбору – це структура даних, яка представляє перетворення абстрактного дерева в дерево, що враховує особливості та правила граматики вихідної мови, але без врахування стилістичних налаштувань. В цьому дереві містяться синтаксичні елементи тексту та їх взаємозв'язки згідно з правил граматики.

На верхньому рівні воно має розбиття на файли, потім рівень просторів імен, далі класи і інтерфейси, рівень методів; рівень аргументів та логіки. Кожний вузол у дереві розбору відображає синтаксичний елемент, як то оператор, вираз, ключове слово. Ребра це відносини між синтаксичними елементами, які визначають ієрархію та порядок вкладеності.

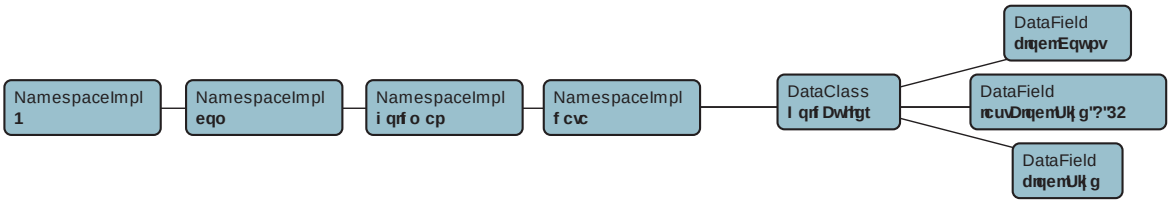

Рис. 2. Абстрактне синтаксичне дерево для структури GoldBuffer

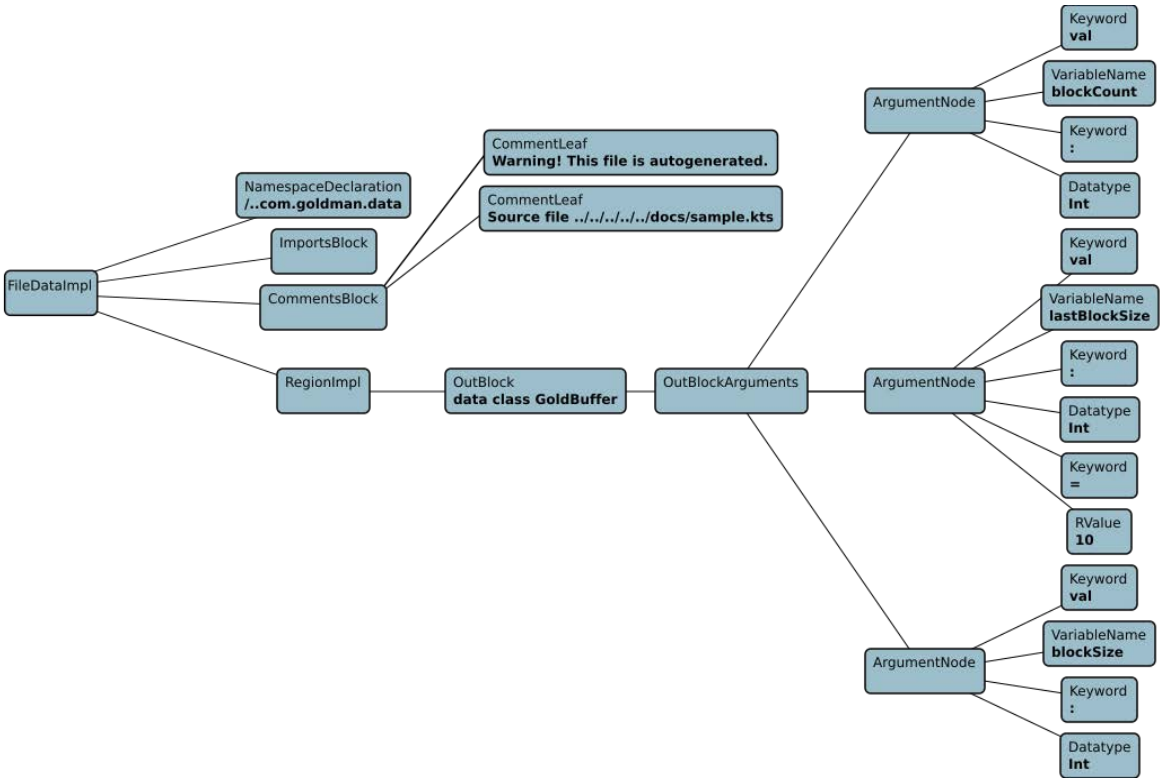

Рис. 3. Дерево розбору для структури GoldBuffer

На рис. 3 наведено дерево розбору для структури GoldBuffer для мови Kotlin. На цьому рівні система вже будує файло-орієнтоване дерево, оскільки на вищому рівні іде вузол FileDataImpl із зазначенням шляху до вихідного файлу. На наступному рівні відбувається: декларування простору імен (NamespaceDeclaration); блок імпортів інших класів (ImportsBlock), в нашому випадку він пустий; блок з коментарями (CommentsBlock), містить автоматично додані коментарі; вузол з описом структури (OutBlock), в свою чергу він містить вхідні аргументи (OutBlockArguments), і далі на нижньому рівні бачимо опис трьох полів даних: blockSize, blockCount, lastBlockSize. Кожне з цих полів описано декількома вузлами дерева.

Дерево розбору не враховує можливі налаштування стандарту оформлення коду на мові високого рівня. Для цього потрібен окремий етап трансформації дерева розбору в стилістичне дерево, яке містить елементи оформлення коду, такі як табуляція, коми, переноси рядків, елементи коментування, інші спеціальні символи. Таким чином стилістичне дерево – це дерево розбору доповнено вузлами оформлення коду.

На рис. 4 приклад того, що буде в результаті трансформації дерева розбору структури

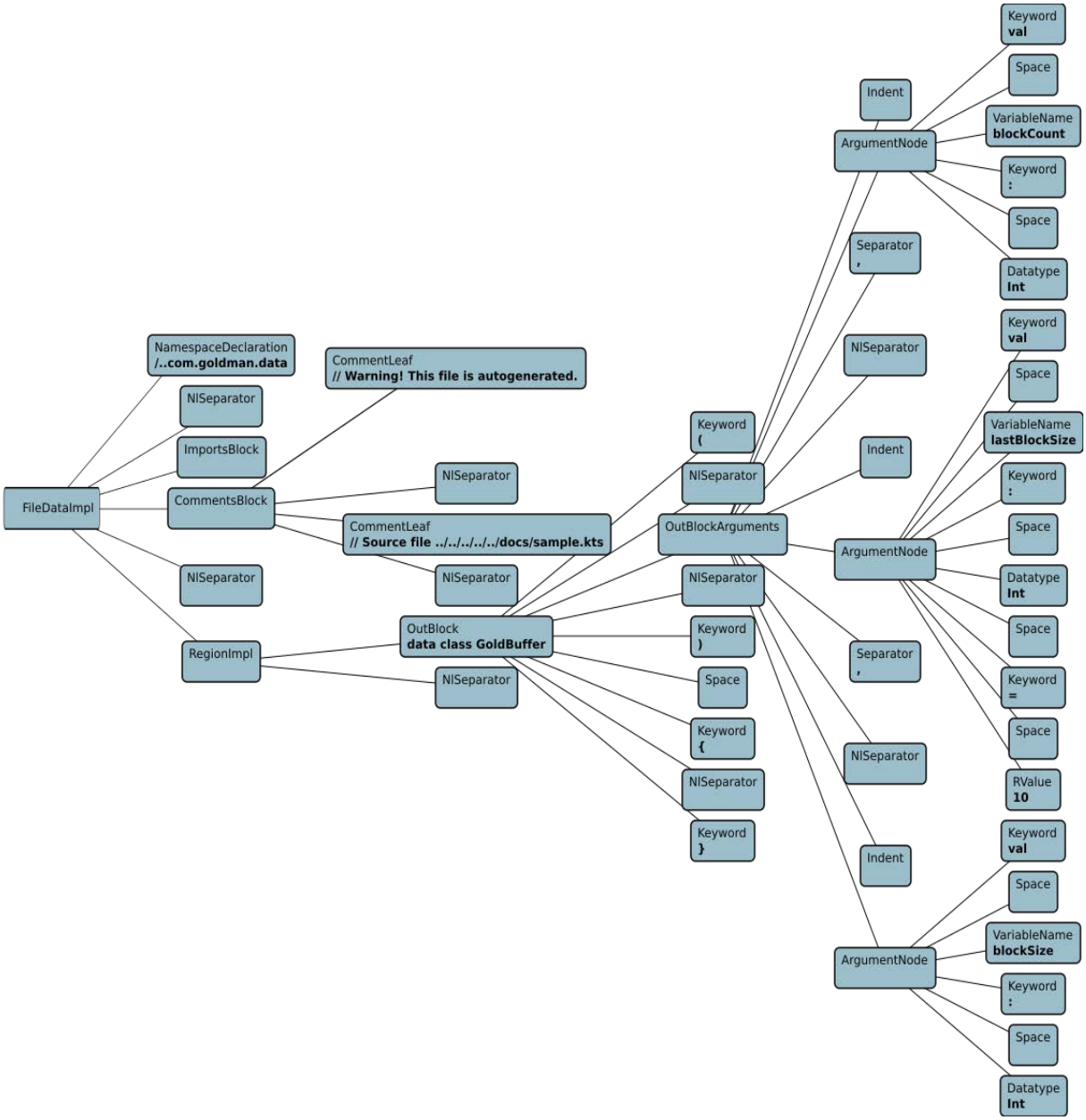

Рис. 4. Стилістичне дерево для структури GoldBuffer

GoldBuffer в стилістичне дерево. Дерево у порівнянні з рис. 3 не змінило своєї структури, а лише доповнилось додатковими вузлами, які відповідають саме за оформлення вихідного файлу.

Розглянемо ці вузли: NlSeparator – відповідає за запис у вихідний файл символу переносу рядку; Space – звичайний символ пробілу – " ", приклад використання – відступ між типом даних та назвою змінної; Indent – символ табуляції, використовується для відступів у вкладених блоках коду; Keyword – вузли з символами "(", ")" для оформлення аргументів функцій, методів та конструкторів; крім того система на цьому етапі розставляє символи дужки для оформлення логічних блоків у місцях, які задекларовані стандартом оформлення коду.

За стилістичним деревом система пише вихідні файли на обраних мовах. В подальшому ці файли можна буде використовувати в проєктах користувача системи, тобто це і є фінальний результат системи кодогенерації. Результуючий файл для нашого прикладу представлено у лістингу 2. Це типова структура з трьома полями на мові Kotlin. Вона отримана за рахунок роботи блоку кодогенерації.

Лістинг 2. Згенерований код структури GoldBuffer

```
1. package com.goldman.data
2. // Warning! This file is autogenerated.
3. // Source file ../../../../../test/dataclass.kts
4.
5. data class GoldBuffer(
6.  val blockCount: Int,
7.  val lastBlockSize: Int = 10,
8.  val blockSize: Int,
9.  ) {
10.}
```

Система кодогенерації розроблена на мові Kotlin з використанням середовища розробки IntelliJ IDEA. Налаштування проєкту можуть бути збережені в JSON файлі або Groovy скрипті.

В якості архітектурного підходу для нашої системи було обрано гексагональну архітектуру (Clean Architecture [9]), яка відповідає нашим вимогам та цілям як найкраще. Гексагональна архітектура – це архітектура, побудована на використанні слабо пов'язаних компонентів. Тобто усі зв'язки між компонентами реалізовані за допомогою адаптерів та інтерфейсів. Використання такої архітектури робить компоненти легко замінними на будь-якому рівні, та полегшує тестування логіки. Цей підхід допомагає забезпечити високу якість коду, зручність підтримки та гнучкість системи для майбутніх змін та розширень. Система використовує XML формат для передачі проміжних результатів між різними етапами виконання генерації.

Були розроблені сценарії для побудови абстрактного синтаксичного дерева, трансляції його в дерево розбору, сценарії для додавання стилістичної інформації та запису в вихідні файли згенерованого коду. Система дозволяє генерувати структури та інтерфейси (примітиви) з мінімальним втручанням людини. Код системи кодогенерації є в публічному доступі, репозиторій з кодом викладено на GitHub за адресою https://github.com/vshcryabets/codegen (дата звернення: 22.01.2024). Кожен може використовувати його в своїх проєктах за ліцензією GPL-3.0 (GNU General Public License version 3). Генерацію коду можливо під'єднати до системи збірки Gradle.

Висновки. Таким чином, було розроблено та запропоновано архітектуру системи кодогенерації. Для подальшого масштабування вона має розділення на різні етапи обробки вхідних описів. Проміжні результати між етапами зберігаються у вигляді дерев, таких як абстрактне синтаксичне дерево, дерево розбору, стилістичне дерево. Було розроблено сценарії для побудови абстрактного синтаксичного дерева, трансляції його в дерево розбору, сценарії для додавання стилістичної інформації та запису в вихідні файли згенерованого коду. На даний час реалізовано кодогенератор, який з опису на метамові може генерувати загальні інтерфейси, блоки констант, перерахування та структури на мовах Kotlin, C/C++ , Java, Swift з мінімальним втручанням людини. Планується додати підтримку інших мов високого рівня, зокрема Rust та Python. Кодогенератор викладено у вільний доступ для ознайомлення на платформі GitHub. Спільнота розробників може запропонувати свої правки та покращення системи за допомогою механізмів пул-реквестів або створення запитів про помилки на GitHub.

Робота над системою продовжується, репозиторій постійно оновлюється, виправляються помилки та система поліпшується. Ведеться робота з удосконалення трансляторів для генерації класів, методів та логіки. Також планується розробка методик взаємодії та поширення коду між командою розробників за допомогою інструментарію git. Система генерації коду під'єднується до системи збірки Gradle.

На даний час кодогенератор використовується для генерації коду у виробничому процесі для суміщення старої та нової частини комерційного продукту, написаного на різних мовах програмування, що допомагає спростити розробку та зекономити час розробників.

Список літератури:

1. Грицюк Ю.І., Рак Т.Є. Програмування мовою С++: навчальний посібник. – Львів: Вид-во Львівського ДУ БЖД, 2011. 292 с.
2. Lua Documentation. URL: https://www.lua.org/docs.html (date of access: 20.01.2024).
3. Eric Windmill. Flutter in Action: Manning Publications, 2020. 368 p.
4. Kotlin Multiplatform. URL: https://kotlinlang.org/docs/multiplatform.html (date of access: 20.01.2024).
5. React Native. URL: https://reactnative.dev/ (date of access: 20.01.2024).
6. What is Xamarin? URL: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (date of access: 20.01.2024).
7. 5 AI Tools That Can Generate Code To Help Programmers. URL: https://www.forbes.com/sites/janakirammsv/2022/03/14/5-ai-tools-that-can-generate-code-to-help-programmers/?sh=7f4865b35ee0 (date of access: 21.01.2024).
8. Шкрябець В.І., Дьячук Т.С. Транскомпілятор як засіб мультиплатформленої розробки. Матеріали XI міжнародної науково-практичної конференції «Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій», Запоріжжя. 2022. С. 119–121.
9. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Pearson; 1nd edition, 2017. 352 p.

Diachuk T.S., Shkriabets V.I. AUTOMATED SYSTEM OF CODE GENERATION IN HIGH-LEVEL LANGUAGES

The article is devoted to solving the practical problem of simplifying the development of applications that must work on many platforms, using an automated code generation system in high-level languages. The field of use of the system is a software development companies. The developed system is part of a more complex machine translation system from one high-level language to another. To achieve the goal, the following was done: an analysis of multi-platform development optimization approaches was carried out, an automatic code generation system was designed, developed and implemented, it was implemented in the production process of a software development company, and the results were analyzed. The object of research is the syntax of high-level programming languages. The subject of research is an automated code generation system. Research methods are based on the syntax of high-level programming languages. To implement the systems, the following are used: Kotlin language for system description and development, Gradle build automation system, XML format for transferring intermediate results between generation execution stages, InteliJ Idea development environment, Git file version control system, GitHub – source code repository. The proposed system architecture of code generation for further scaling includes a division into different stages of processing input descriptions. Intermediate results between steps are stored as trees, such as abstract syntax tree, parsing tree, stylistic tree. As a result, the user receives the output files with the generated code. During the designing the system, a metalanguage was developed, which contains methods for declaratively describing structures, interfaces, data types, and methods. The proposed code generation system from description to metalanguages can generate structures and interfaces (primitives) in Kotlin, C/C++, Java, Swift languages with minimal human intervention. The code generator is published on the GitHub platform. Anyone can use the developed system in their projects under the GPL-3.0 license and suggest their edits and improvements using the pull request mechanisms or by creating bug requests on GitHub.

Key words: *github, kotlin, meta language, abstract syntax tree, parse tree, code generation, OS, stylist tree.*