УДК 004.92 DOI https://doi.org/10.35546/kntu2078-4481.2025.1.2.5

В. М. ВАСИЛИК
студент-магістр кафедри інженерії програмного забезпечення
Івано-Франківський національний технічний університет нафти і газу
ORCID: 0009-0007-7676-5316

Л. М. ГОБИР
асистент кафедри інженерії програмного забезпечення
Івано-Франківський національний технічний університет нафти і газу
ORCID: 0009-0007-3176-2314

Т. О. ВАВРИК
асистент кафедри інженерії програмного забезпечення
Івано-Франківський національний технічний університет нафти і газу
ORCID: 0000-0002-0612-0084

ТЕОРЕТИЧНІ АСПЕКТИ ВІЗУАЛЬНИХ МОВ ПРОГРАМУВАННЯ

Візуальні мови програмування не є широко розповсюдженими для розробки складного програмного забезпечення. Це стаття є теоретичною частиною комплексного дослідження, що має дві основні цілі. По-перше, виявити, як візуальні мови програмування впливають на гнучкість програмного забезпечення. По-друге, визначити, чи мають вони переваги при розробці програмного забезпечення з когнітивними обмеженнями (людська пам'ять, концентрація, увага, навігація, швидкість мислення тощо). Результати цього дослідження є важливими для практичної частини. В статті розглянуто п'ять сучасних візуальних мов програмування, зроблено огляд двох оглядово-аналітичних статей про візуальні мови програмування та проаналізовано десять наукових праць, які практично використовують об'єкт дослідження у своїй прикладній галузі. На основі зібраного матеріалу були створені принципи візуальних мов програмування, що покращують гнучкість програмного забезпечення. Відповідно до цих принципів була створена архітектура. Щоб бути ефективним інструментом розробки, потрібне спеціалізоване середовище. Дослідження також надає рекомендації щодо впровадження інтегрованого середовища розробки. Створені принципи мають певний позитивний вплив на архітектуру, і для того, щоб перевірено стверджувати, що програмне забезпечення стає більш гнучким, необхідні подальші практичні дослідження. Подальші дослідження також повинні включати когнітивні та психологічні тести, щоб стверджувати про перевагу над текстовими мовами програмування загального призначення. Таким чином, можна тільки припустити, що принципи, архітектура та концепція інтегрованого середовища розробки для візуальних мов програмування, що викладені в статті, мають тенденцію до зменшення впливу когнітивних факторів на процес розробки програмного забезпечення. Теоретичні результати цього дослідження необхідні для того, щоб візуальні мови програмування стали настільки ж широко застосовними, як і текстові мови програмування загального призначення.

Ключові слова: *архітектура, візуальне програмування, візуальна мова програмування, візуально-орієнтоване програмування, інтегроване середовище розробки.*

V. M. VASYLYK
Master's Student at the Department of Software Engineering
Ivano-Frankivsk National Technical University of Oil and Gas
ORCID: 0009-0007-7676-5316

L. M. HOBYR
Assistant at the Department of Software Engineering
Ivano-Frankivsk National Technical University of Oil and Gas
ORCID: 0009-0007-3176-2314

T. O. VAVRYK
Assistant at the Department of Software Engineering
Ivano-Frankivsk National Technical University of Oil and Gas
ORCID: 0000-0002-0612-0084

THEORETICAL ASPECTS OF VISUAL PROGRAMMING LANGUAGES

Visual programming languages are not widely used for developing complex software. This paper is a theoretical part of a complex study with two main goals. First, to identify how visual programming languages affect software flexibility. Second, to determine whether they have advantages when developing software with cognitive limitations (human memory,

concentration, attention, navigation, speed of thinking, etc). The results of this study are important for the practical part. The article considers five modern visual programming languages, examines two review and analytical articles about visual programming languages, and analyzes ten scientific papers that practically use the object of study in their applied field. Based on the collected material, the principles of visual programming languages were created to improve software flexibility. An architecture was created in accordance with these principles. To be an effective development tool, a specialized environment is required. The study also provides recommendations for implementing a visual integrated development environment. The established principles have a certain positive impact on the architecture, and further practical research is needed to reliably assert that software is becoming more flexible. Further research should also include cognitive and psychological tests to claim superiority over general-purpose text-based programming languages. Thus, it can only be assumed that the principles, architecture and concept of an integrated development environment for visual programming languages outlined in this paper tend to reduce the influence of cognitive factors on the software development process. The theoretical results of this study are necessary for visual programming languages to become as widely used as general-purpose textual programming languages.

Key words: *architecture, visual programming, visual programming language, visual-oriented programming, integrated development environment.*

Постановка проблеми

Розробники можуть підвищити швидкість проєктування та якість ПЗ за допомогою зручних інструментів та концепції «якість-ціна-час». Складну структуру кодів важко утримувати в людській пам'яті через когнітивні обмеження, тому потрібна велика кількість документації. Візуальна розробка зменшує цю потребу, слугуючи самостійною документацією. Важливо створити вищий рівень абстракції для маніпулювання архітектурою ПЗ. Візуальні мови програмування (VPLs) забезпечують абстракцію над кодом, покращують розуміння предметної області через моделювання архітектури як семантичної мережі. У таких моделях вузли (поняття) та ребра (зв'язки) полегшують навігацію та маніпулювання складними структурами. Поєднання VPLs з об'єктно-орієнтованим підходом створює потужний інструмент для моделювання та розробки ПЗ. Існують рішення для перетворення UML у програмний код, хоча вони мають обмеження. Когнітивні обмеження при роботі з текстовим кодом спричиняють повільну обробку та помилки. Візуальні елементи швидше обробляються мозком людини. Програмний код часто має синтаксичні помилки, тоді як візуальні елементи можуть мати тільки логічні помилки, які легко виправити. Блок-схеми сприяють реалізації алгоритмів і зменшують ймовірність помилок. Сучасні технології дозволяють автоматично перетворювати блок-схеми у програмний код. VPLs в поєднанні з компонентно-орієнтованим підходом підвищують гнучкість ПЗ. Візуальні мови програмування часто використовуються в освітніх цілях через свої переваги. Вони перекладаються в код без синтаксичних помилок і мають описи та підказки природною мовою. Інтегровані середовища візуальної розробки забезпечують високу зручність і доступність, швидку розробку та створення прототипів. Традиційне програмування має кілька бар'єрів для початківців. Якщо початківець не розуміє синтаксису, він зіткнеться з дрібними помилками, такими як зайві або пропущені дужки та знаки пунктуації. Якщо він не розуміє семантики, то не буде розуміти поведінку операцій та функцій. Якщо він не розуміє прагматики, то не зрозуміє, коли і як використовувати операції та функції [2, с. 288]. Однак сучасні VPLs менш потужні, ніж текстові мови, і часто не підтримують структури масивів, багатопотоковість, модульний чи об'єктно-орієнтований підхід, а також роботу з файловою системою [9, с. 55]. Підхід візуального програмування використовується в доменно-специфічних мовах (DSL), які спеціалізуються на конкретних завданнях [6, с. 3], платформах малокодової розробки (LCDP), що дозволяють проєктувати ПЗ за допомогою візуальних інструментів, і в інженерії, керованій моделями (MDE), де моделі використовуються для всієї розробки та підтримки ПЗ [12, с. 440]. Складність програмного забезпечення зростає з появою нових технологій, що створює виклики для чистого візуального програмування.

Аналіз останніх досліджень і публікацій

У галузі інженерії та розробки програмного забезпечення існує безліч підходів, методологій, технік, інструментів та теорій. Для створення нового чи вдосконалення продукту, що існує використовується наявна теоретична база. У ході пошуку та аналізу сучасних візуальних мов програмування було враховано дані з досліджень [9, 14]. Критерії відбору VPLs включали наявність оновлень протягом останніх двох років, наявність персонального вебсайту, спеціалізованого середовища розробки та сучасного дизайну інтерфейсу користувача. Серед 53 VPLs, описаних в аналітичних статтях, було відібрано 5. Складено аналітичну та порівняльну таблицю. Наступним кроком був аналіз статей у наукових базах даних за такими критеріями пошуку: ключові слова (Visual language, visual programming, visual programming language, visual programming environment) та наукові бази даних (ScienceDirect, ResearchGate, Google Scholar, Wikipedia). Загалом було відібрано 10 наукових статей з різних сфер застосування, щоб зробити наведені ідеї більш застосовними.

Отримані результати аналізу не мають на меті порівняння кандидатів на звання найкращого VPL, вони є лише інформативними. Дивлячись на доступні функціональні можливості можна визначити, що є загальноприйнятою практикою для візуальних мов програмування. Raptor – це середовище програмування на основі

блок-схем розроблене для допомоги студентам у візуалізації алгоритмів та уникненні синтаксичної складності. Flowgorithm – це безплатна мова програмування для початківців, яка базується на графічних блок-схемах. Scratch функціонує як вебрішення з офлайн-редактором для створення ігор, анімації та історій. App Inventor – це візуальне середовище програмування дозволяє користувачам, зокрема програмістам-початківцям, а також викладачам, створювати мобільні застосунки для пристроїв Android та iOS за допомогою інтерфейсу drag-and-drop. Node-RED – це інструмент для візуального програмування на основі потоку з низьким вмістом коду розроблений для підключення апаратного забезпечення, API та онлайн-сервісів у проєктах IoT.

Після ретельного дослідження було інстальовано та досліджено IDE для кожного з вищезгаданих VPLs. У таблиці 1 наведені результати дослідження. Розглядалися лише загальні функції доступні кожному. Кожне середовище розробки може мати свої особливості, які потребують глибшого дослідження.

Raptor з його невеликим набором візуальних блоків все ще може бути хорошим програмним забезпеченням для навчання через його простоту та візуалізацію виконання програми. Flowgorithm є привабливим через широку підтримку GPL та багатомовний інтерфейс, а також чудові навчальні ресурси. Scratch, орієнтований на конкретну область і маючи інтуїтивно зрозумілий інтерфейс навіть для дітей, займає міцні позиції серед освітніх програм. Попри переваги інтерфейсу перетягування та скидання, інтуїтивно зрозумілої блокової логіки, App Inventor не використовується для створення складного програмного забезпечення, причиною може бути слабка підтримка проєкту з боку платформ Andoroid та iOS. Node-RED використовується технологічними компаніями (дані з їх вебсайту), він побудований на Node.js, який дає доступ до багатьох бібліотек, які можна використовувати у функціональних блоках, одним із факторів його ефективності є поєднання візуальних об’єктів та JavaScript-функцій. Щоб візуальні мови програмування використовувалися не тільки для вузькоспеціалізованих задач, вони повинні мати

Таблиця 1

Аналітичне порівняння VPLs та їх середовищ

№	Назва	Сфера застосування	Supported GPL	Ліцензія	Базується на	Сильні сторони та унікальні особливості
1	Raptor	Освіта	C, Java, Ada, C# та VBA	GNU General Public License	Блок-схема	Модульний та об’єктно-орієнтований; інструменти налагодження
2	Flowgorithm	Програмна інженерія	C++, C#, Java, Python і ще 10	Freeware	Блок-схема	Користувачі можуть контролювати швидкість виконання блок-схем; підтримує декілька мов програмування
3	Scratch	Освіта	Не підтримує	MIT	Блок	Доступний як вебплатформа, а також пропонує офлайн-редактор; спеціалізується на інтерактивних історіях, іграх та анімації
4	App Inventor	Мобільна розробка	Java	MIT	Блок	Підтримка емуляторів; інтерфейс drag-and-drop
5	Node-RED	Інтернет речей	JavaScript	Apache 2.0	Блок-схема	Підтримка інтернет-протоколів; програмування потоком даних

Таблиця 2

Використання VPLs

№	Назва статті / Висновок після огляду
1	Raptor: a visual programming environment for teaching object-oriented programming [13] / Raptor – це візуальне середовище програмування, призначене для навчання об’єктно-орієнтованого програмування. Воно активно застосовується в освітній сфері та підтримує основні принципи об’єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм. UML дизайнер Raptor дозволяє користувачам створювати класи, інтерфейси та типи перерахувань. Крім того, середовище має понад 80 вбудованих функцій і процедур, які дозволяють студентам генерувати випадкові числа, виконувати тригонометричні обчислення, малювати графіку (кола, квадрати, лінії тощо) та взаємодіяти з вказівними пристроями. Raptor використовує структуровану методику навчання, що дозволяє студентам виконувати програму покроково або запускати її безперервно. Коментування в Raptor здійснюється шляхом натискання правої кнопки миші на символі та вибору пункту «коментар»
2	A visual programming environment for functional languages [11] / Візуальне середовище програмування для функціональних мов застосовується в галузі комп’ютерних наук. Воно описує методи перенесення елементів функціонального програмування у візуальне середовище, розглядає питання створення налагоджувачів для візуальних середовищ програмування та перевірки синтаксису візуальних об’єктів. У роботі наведено приклади успішних практик, що демонструють реалізацію візуального функціонального середовища програмування (VFPE)
3	A didactic object-oriented, prototype-based visual programming environment [7] / Візуальне середовище програмування для дидактичних цілей, що поєднує об’єктно-орієнтовану та прототипно-базову моделі, використовується в освітній сфері. У ньому представлено синтаксис візуальної мови програмування, що сприяє кращому засвоєнню матеріалу студентами. Серед передових практик відзначено поєднання прототипно-базової моделі з об’єктно-орієнтованою моделлю, що дозволяє більш ефективно навчати основних концепцій програмування
4	JSPatcher, a visual programming environment for building high-performance web audio applications [10] / JSPatcher – це візуальне середовище програмування, призначене для створення високопродуктивних вебаудіо застосунків. Сфера застосування охоплює обробку мультимедійних даних. У середовищі JSPatcher представлено архітектуру та синтаксис візуальної мови програмування, що полегшує роботу з аудіоданими. До передових практик відноситься використання інтерфейсу з батьківської загальної мови програмування, що забезпечує високу продуктивність та гнучкість при розробці аудіозастосунків

широку базу бібліотек загального призначення. Постійні оновлення є критично важливими для VPL, оскільки без них клієнти втрачають довіру до технології, а її місце займають більш просунуті рішення. У наступних статтях [1, 3, 4, 5, 7, 8, 10, 11, 13, 15] надано інформацію про те, як VPLs застосовуються в різних сферах. Таблиця 2 демонструє їх досвід.

Існує багато способів розробки та впровадження VPLs та їх середовищ. Поєднання декількох моделей програмування та підходів робить цей процес надійнішим і позитивно впливає на VPL. Матеріали дослідження візуальних мов програмування та огляд наукових статей значно покращили результати цього дослідження, які викладено далі.

Формулювання мети дослідження

Метою дослідження є створення теоретичної бази для візуальних мов програмування. Завдання полягають у розробці принципів візуальних мов програмування та їх архітектури, створені пропозиції для їх інтегрованого середовища розробки. Задум комплексного дослідження також полягає в створенні пропозицій щодо розробки робочого середовища на основі візуальних мов програмування, яке зменшує вплив когнітивних та біологічних обмежень, таких як здатність людини обробляти інформацію, зберігати її в пам'яті та фокусувати увагу на певних завданнях.

Викладення основного матеріалу дослідження

Це дослідження сформувало принципи VPL. Принцип 1. Парадигми програмування, моделі, підходи, алгоритми та структури даних, розроблені в GPL, можуть бути використані у VPL з певним рівнем абстракції. Принцип 2. Використання коду як документації. Оскільки у VPL немає коду, візуальні елементи є одночасно абстракцією коду та документацією. Принцип 3. Кожен рівень архітектури поділяється на домен, який може бути розбитий на менші частини та мати підрівні шляхом декомпозиції. Принцип 4. Перегляд дочірнього контенту та батьківських зв'язків на одному рівні є виснажливим для розробника, тому вони повинні бути розділені. Принцип 5. VPL використовують компонентний підхід, що забезпечує модульність програмного забезпечення, можливість декомпозиції на менші компоненти, повторне використання коду, поширення шаблонів і бібліотек. Принцип 6. VPL використовують об'єктно-орієнтований підхід. Класи не містять реалізації даних, а лише посилання на схему доменів даних, методи яких можуть отримувати доступ до даних. Класи не містять реалізацій методів, а лише посилання на методи, що знаходяться в домені методів. Принцип 7. Поліморфізм у VPL реалізований таким чином, що не клас визначає реалізацію методу, а метод при виклику залежно від типу об'єкта в якому він був викликаний, визначає спосіб своєї поведінки. Принцип 8. Класи також мають поліморфізм для схеми даних, вони посилаються на схему даних із домену даних, де визначається їх обсяг, тип та обмеження залежно від типу об'єкта, який до них звертається. Принцип 9. VPL повинна мати доступ до батьківських бібліотек обраної GPL, а також створювати свої власні обгортки на їх основі. Принцип 10. Структура пакунків розглядається не як лінійна структура тек, а як двовимірна область певного домену. Вона може бути представлена графом, де кожна вершина може приховувати новий підграф. Принцип 11. VPL повинні використовувати елементи діаграм варіантів використання (Use Case) з UML. Актори мають візуальне представлення, а також взаємозв'язок один з одним, прецеденти наведені окремо у формі таблиць. Принцип 12. Кожен елемент повинен мати можливість деактивуватися і ставати недоступним для взаємодії з іншими. Деактивація об'єктів корисна для налагодження, а також для заміни старих реалізацій на нові, сприяючи швидкій розробці ПЗ. Принцип 13. VLP повинні підтримувати версійність програмного забезпечення та можливість відновлення попередніх версій. Принцип 14. Абстракції у VPL не вимагають генерування того самого коду для батьківської текстової мови програмування. Наприклад, об'єктно-орієнтовані абстракції можна перетворити на процедурний код. Принцип 15. Кожен клас повинен бути пов'язаний з іншими класами через семантичну модель, де вузли відповідають класам, а відносини між ними представляються прецедентами їх методів. Переваги VPL та гнучкість програмного забезпечення залежать від обраної архітектури. Створена архітектура (рис. 1) має шість підгруп, що відповідають за певні функції в програмному забезпеченні. Серед них: варіанти використання, інтерфейс користувача, дані, потоки, поліморфізм класів, відносини класів.

UML діаграма варіантів використання є інструментом для візуалізації вимог до програмного забезпечення, зосередженим на акторах та прецедентах. Актори представляють учасників системи з різними дозволами, а прецеденти відображають дії, які ці актори можуть виконувати. Для забезпечення гнучкості архітектура поєднує зв'язки між акторами та дозволи, і прецеденти, що є абстракціями для методів. Наприклад, користувач має дозвіл на читання файлу, тому він має варіант використання «Читання файлу», а метод надасть програмну логіку для читання файлу.

Існують різні способи створення користувацького інтерфейсу, але для гнучкості програмного забезпечення важливим є слабкий зв'язок. Архітектура вказує, як забезпечити слабкий зв'язок між інтерфейсом і логікою програми. Метод викликає подію, яка змінює стан, після чого інтерфейс користувача змінюється відповідно до заданих умов. Це означає, що навіть при зміні логіки програми користувацький інтерфейс не потребує змін.

Згідно з принципами для VPL, архітектура підкреслює поліморфізм. Класи можуть бути пов'язані між собою і містити посилання на домени даних і методів замість безпосередніх даних або методів. Класи поділяються на два типи: одні реалізують логіку програмного забезпечення, інші – акторів. Класи з логікою ПЗ не повинні бути пов'язані з акторами.

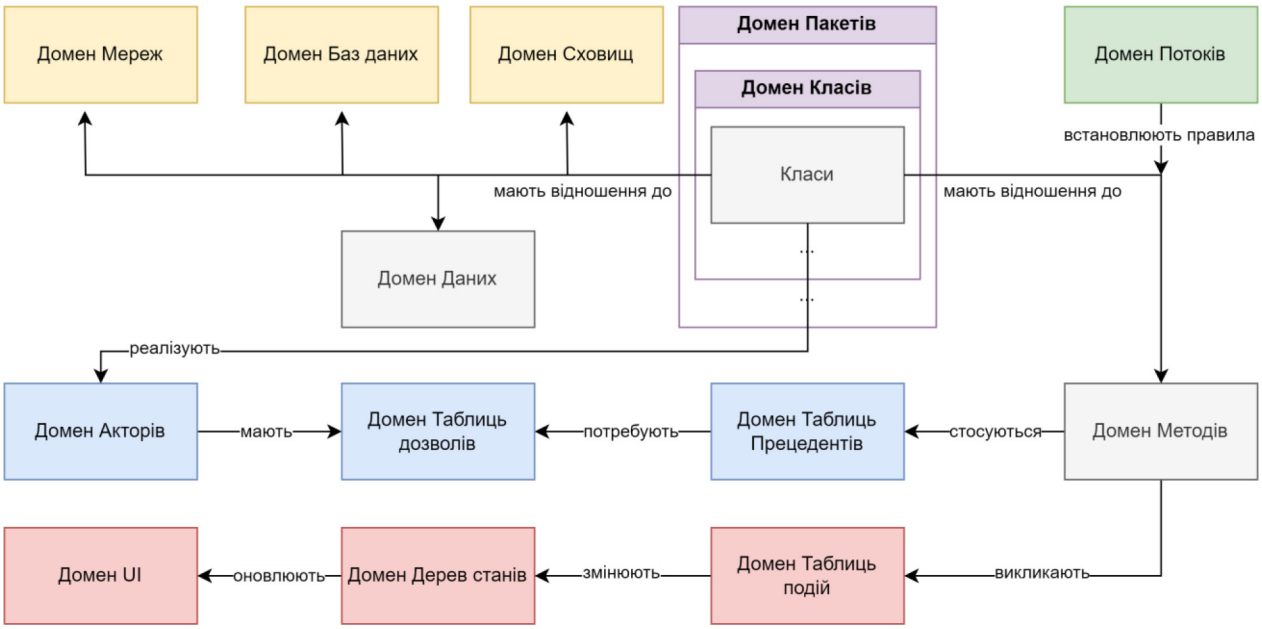

Рис. 1. Архітектура для VPLs

Архітектура потоків важлива для складних систем, які можуть заплутати розробника своєю складністю. Клас викликає метод, а домен потоків визначає, на якому потоці його виконувати. Потоки пов'язані з поняттями пулу потоків і ядер процесора та мають візуальне представлення у VPL. Програмне забезпечення може виконувати обчислення на віддалених серверах. Домен потоків може змінювати поведінку системи залежно від умов, наприклад, використання віртуального сервера при нестачі локальних ресурсів. VPL допомагають розробнику бачити всі доступні ресурси.

Дані є важливою частиною класів. Домен мережі визначає, до яких мереж може отримати доступ програма, забезпечуючи безпеку та обмежуючи джерела даних. Складне ПЗ може використовувати кілька баз даних і мати умови, що змінюють його поведінку. У VPL важливо візуалізувати всіх учасників, які зберігають або надають дані. Домен сховищ відповідає за роботу з файлами: читання, редагування та створення. Програма може мати доступ до декількох дисків операційної системи або хмарного сховища. Візуалізація всіх учасників роботи з даними є важливою частиною VPL.

Остання підгрупа відповідає за пакети та класи. Зв'язки між класами створюються через предметну область або специфічну архітектуру ПЗ. Візуальне представлення зв'язків між класами у VPLs повинно відповідати принципам. Кожен проєкт має кореневий пакет, який може містити домени. Кожен домен пакета може містити внутрішні пакети та класи або їхні домени. Домен класу може містити лише класи. Клас доменів визначає їх спільну категорію. Домен пакета – це категорія для внутрішнього вмісту.

Перевагою цієї архітектури є легкість перевірки синтаксису, оскільки її елементи мають мало зв'язків між собою. Частини архітектури розташовані за лінійним шаблоном і мають лише двох сусідів, тому програмне забезпечення набуває гнучкості. Використання такої архітектури в GPL створить багато зайвого коду і вимагало б від розробника великої кількості операцій, що ускладнює керування об'єктами. У VPL дана архітектура не впливає на швидкість розробки, оскільки всі операції виконує середовище, яке пропонує багато форм і візуальних просторів для зручного маніпулювання всіма елементами архітектури.

Розроблені принципи VPL тісно пов'язані з середовищем розробки в якому вони можуть бути застосовані. І запропонована архітектура досягає своєї гнучкості та швидкості розробки лише у відповідному інтегрованому середовищі розробки. Для створення інтегрованого середовища розробки до візуальних мов програмування варто передбачити зручні інструменти для створення інтерфейсів користувача, як декларативні, так і drag-and-drop. Використання нейронних мереж допоможе у створенні розмітки інтерфейсу з природної мови. Інтерфейс перетягування візуальних компонентів у робочу область проєкту та автоматична перевірка семантичних моделей значно спростять процес розробки. Середовище повинно підтримувати різні парадигми програмування генеруючи відповідний код, і забезпечувати перевірку синтаксису в режимі реального часу. Інструменти для налагодження є обов'язковими, а кожен рівень архітектури повинен мати окрему вкладку. Підтримка open source шаблонів і сучасних методологій розробки також є важливою. Необхідно реалізувати можливість окремого версіювання для кожного рівня та всього проєкту загалом, а також інтегрувати бібліотеки створені під GPL у візуальні конструкції.

Важливо забезпечити можливість розширення списку мов програмування іншими розробниками, підтримку діаграм та візуалізацію даних.

Висновки

Ця стаття буде корисною для дослідників та розробників, які хочуть працювати з візуальними мовами програмування та їхніми інтегрованими середовищами розробки. Вона не охоплює всі складнощі розробки та впровадження IDE для VPL, але принципи, що забезпечують гнучкість програмного забезпечення, можуть бути використані поза контекстом візуальних мов програмування. Для цього дослідження зібрали аналітичні та інженерні матеріали й створили два десятки принципів для VPLs, які сприятимуть їхньому розвитку. У науковій роботі аналізуються сучасні VPLs та дослідження в цій галузі, описуються принципи реалізації візуальної мови програмування і пропозиції щодо специфікацій, які повинні мати їх інтегровані середовища розробки. Також представлено нову архітектуру, яка тісно пов'язана з цими принципами. Ідеї запропоновані в статті можуть бути неповними та потребують подальших досліджень. Було використано обмежену кількість аналітичних статей про VPLs та проведено аналіз декількох IDEs, тому дані можуть бути неповними. Ідеї щодо розробки VPL загального призначення не мають фінансового обґрунтування і потребують спочатку напрацювання теоретичної бази та інноваційних підходів, щоб обґрунтувати ризики. Впровадження описаної архітектури VPL та її IDE концепції може мати позитивні результати, і автори планують продовжувати дослідження VPLs та практичне застосування розробленої теорії.

Список використаної літератури

1. Aleshchenko O. Visual programming system. *ICSFTI2020* : Міжнар. науково-техн. конф., м. Київ, 13–14 трав. 2020 р. 2020. С. 185–191.

2. A programming environment for visual block-based domain-specific languages / A. Kurihara та ін. *Procedia computer science*. 2015. Т. 62. С. 287–296. URL: https://doi.org/10.1016/j.procs.2015.08.452 (дата звернення: 13.12.2024).

3. A use-case for behavioral programming: An architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios / A. Ashrov та ін. *Science of computer programming*. 2015. Т. 98. С. 268–292. URL: https://doi.org/10.1016/j.scico.2014.01.017 (дата звернення: 13.12.2024).

4. A visual programming environment for learning distributed programming / B. Broll та ін. *SIGCSE '17* : Техн. симп. 2017. С. 81–86.

5. Banyasad O., Cox P. T. Visual programming of subsumption-based reactive behaviour. *International journal of advanced robotic systems*. 2008. Т. 5, № 4. С. 42. URL: https://doi.org/10.5772/6226 (дата звернення: 13.12.2024).

6. Galhardo P., Silva A. R. d. Combining rigorous requirements specifications with low-code platforms to rapid development software business applications. *Applied sciences*. 2022. Т. 12, № 19. С. 9556. URL: https://doi.org/10.3390/app12199556 (дата звернення: 13.12.2024).

7. García Perez-Schofield B., Ortin F. A didactic object-oriented, prototype-based visual programming environment. *Science of computer programming*. 2019. Т. 176. С. 1–13. URL: https://doi.org/10.1016/j.scico.2019.02.004 (дата звернення: 13.12.2024).

8. Grigorenko P., Saabas A., Tyugu E. COCOVILA – compiler-compiler for visual languages. *Electronic notes in theoretical computer science*. 2005. Т. 141, № 4. С. 137–142. URL: https://doi.org/10.1016/j.entcs.2005.05.009 (дата звернення: 13.12.2024).

9. Idrees M., Aslam F. A comprehensive survey and analysis of diverse visual programming languages. *VFAST transactions on software engineering*. 2022. Т. 10, № 2. С. 47–60. URL: https://doi.org/10.21015/vtse.v10i2.1009 (дата звернення: 13.12.2024).

10. JSPatcher, a visual programming environment for building high-performance web audio applications / S. Ren та ін. *Journal of the audio engineering society*. 2022. Т. 70, № 11. С. 938–950. URL: https://doi.org/10.17743/jaes.2022.0056 (дата звернення: 13.12.2024).

11. Kelso J. A visual programming environment for functional languages : thesis. 2002. URL: https://researchrepository.murdoch.edu.au/id/eprint/50254/ (дата звернення: 13.12.2024).

12. Low-code development and model-driven engineering: two sides of the same coin? / D. Di Ruscio та ін. *Software and systems modeling*. 2022. Т. 21, № 2. С. 437–446. URL: https://doi.org/10.1007/s10270-021-00970-2 (дата звернення: 13.12.2024).

13. Martin C. Raptor: a visual programming environment for teaching object-oriented programming.*Journal of Computing Sciences in Colleges*. 2009. Т. 24, № 4. С. 275–281. URL: https://dl.acm.org/doi/abs/10.5555/1516546.1516591 (дата звернення: 11.11.2024).

14. Ray P. P. A survey on visual programming languages in internet of things. *Scientific programming*. 2017. Т. 2017. С. 1–6. URL: https://doi.org/10.1155/2017/1231430 (дата звернення: 13.12.2024).

15. Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review / E. Coronado та ін. *Journal of computer languages*. 2020. Т. 58. С. 100970. URL: https://doi.org/10.1016/j.cola.2020.100970 (дата звернення: 13.12.2024).

References

1. Aleshchenko O. (2020). Visual programming system. *ICSFTI2020 : Mizhnarodna naukovo-tekhnichna konferentsiia* pp. 185–191.

2. Kurihara A. and others (2015). A programming environment for visual block-based domain-specific languages. *Procedia Computer Science*, ISSN: 1877-0509, Vol: 62, pp. 287-296 URL: https://doi.org/10.1016/j.procs.2015.08.452

3. Ashrov A. and others (2015). A use-case for behavioral programming: An architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios. *Science of computer programming*, Vol. 98, pp. 268–292. URL: https://doi.org/10.1016/j.scico.2014.01.017

4. Broll B. and others (2017). A visual programming environment for learning distributed programming. *SIGCSE '17 : Tekhnichnyi sympozium*, pp. 81–86.

5. Banyasad O., Cox P.T. (2008). Visual programming of subsumption-based reactive behaviour. *International journal of advanced robotic systems*, Vol. 5, No 4, pp. 42. URL: https://doi.org/10.5772/6226

6. Galhardo, P., & Silva, A. R. d. (2022). Combining Rigorous Requirements Specifications with Low-Code Platforms to Rapid Development Software Business Applications. *Applied Sciences*, *12*(19), 9556. https://doi.org/10.3390/app12199556

7. García Perez-Schofield B., Ortin F. (2019). A didactic object-oriented, prototype-based visual programming environment. *Science of computer programming*, Vol. 176, pp. 1–13. https://doi.org/10.1016/j.scico.2019.02.004

8. Grigorenko P., Saabas A., Tyugu E. (2005). COCOVILA – compiler-compiler for visual languages. *Electronic notes in theoretical computer science*, Vol. 141, No 4. pp. 137–142. https://doi.org/10.1016/j.entcs.2005.05.009

9. Idrees, M., & Aslam, F. (2022). A Comprehensive Survey and Analysis of Diverse Visual Programming Languages. *VFAST Transactions on Software Engineering*, *10*(2), 47–60. https://doi.org/10.21015/vtse.v10i2.1009

10. S. Ren and others (2022). JSPatcher, a visual programming environment for building high-performance web audio applications. *Journal of the audio engineering society*, vol. 70, No 11, pp. 938–950. https://doi.org/10.17743/jaes.2022.0056

11. Kelso, J. (2002). *A visual programming environment for functional languages* [Murdoch University]. https://researchportal.murdoch.edu.au/esploro/outputs/doctoral/A-visual-programming-environment-for-functional/991005541814107891#file-0

12. D. Di Ruscio and others (2022). Low-code development and model-driven engineering: two sides of the same coin? *Software and systems modeling*, vol. 21(2), pp. 437–446. https://doi.org/10.1007/s10270-021-00970-2

13. Martin C. Carlisle. 2009. Raptor: a visual programming environment for teaching object-oriented programming. *J. Comput. Sci. Coll*. 24, 4 (April 2009), 275–281. https://dl.acm.org/doi/abs/10.5555/1516546.1516591

14. Ray, Partha Pratim, A Survey on Visual Programming Languages in Internet of Things, *Scientific Programming*, 2017, 1231430, 6 pages, 2017. https://doi.org/10.1155/2017/1231430

15. Coronado E., Mastrogiovanni F., Indurkhya B., Venture G. (2020). Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review. *Journal of computer languages*, vol. 58, pp. 100970. https://doi.org/10.1016/j.cola.2020.100970