

ВІЗУАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

¹ Вінницький національний технічний університет;

Анотація

У роботі проаналізовано особливості використання спеціалізованих діаграм і програмних засобів для візуалізації особливостей програмного коду.

Ключові слова: програмне забезпечення, UML, структура програми, динаміка поведінки програми, профілювання, теплові карти коду, аналіз залежностей, автоматичний аналіз коду.

Abstract

In the work, it is analyzed the key aspects of specialized diagrams and software tools usage for visualizing the features of program code.

Keywords: software, UML, program structure, program behavior dynamics, profiling, code heat maps, dependency analysis, automatic code analysis.

Вступ

Подання даних у графічній формі сприяє підвищенню інформативності пропускнуго каналу [1] зв'язку між людиною і комп'ютером. Відповідно, візуалізація даних забезпечує більш глибокий і повний аналіз інформації. Сучасні програмні засоби характеризуються значним обсягом коду, великою кількістю програмних компонент. Тому, для ефективного аналізу характеристик програмного продукту необхідною є візуалізація його основних особливостей.

Метою роботи є проаналізувати основні засоби, що застосовуються для візуалізації особливостей програмного забезпечення.

Результати дослідження

Серед напрямків візуалізації програмного забезпечення (ПЗ) можна виділити візуалізацію структури коду, візуалізацію динаміки виконання програми, візуалізацію прогресу процесу розробки, візуалізацію характеристик продуктивності коду.

Для візуалізації структури програм може бути використаний ряд діаграм UML [2] (Unified Modeling Language) – діаграми класів, компонентів, розгортання, об'єктів, пакунків, композиційної структури, профілів. Діаграми класів відображають типи об'єктів програми, а також статичні зв'язки між ними (асоціація, агрегація, композиція, наслідування). Діаграми компонентів подають взаємозв'язки між програмними компонентами (наприклад, виконуваними файлами). Діаграми розгортання використовуються для подання розподілу артефактів (фізичних одиниць ПЗ) між пристроями. Діаграми об'єктів, на відміну від діаграм класів, відображають відношення між екземплярами класів у визначений момент часу. Діаграми пакунків застосовуються для відображення зв'язків між різними програмними підсистемами та модулями (пакунками). Діаграми композиційної структури призначені для візуалізації статичних відношень між внутрішніми елементами класів. Діаграми профілів призначені для розширення стандартних можливостей UML шляхом введення примітивів нових типів (стереотипів), допоміжних атрибутів (тегованих значень), обмежень.

Для візуалізації динаміки виконання програм також може бути застосований набір діаграм UML [2]: діаграми варіантів використання, діяльності, станів, послідовності, комунікації, взаємодії, синхронізації. Діаграми варіантів використання призначені для візуалізації прикладів функціонування програмного засобу у різному контексті. Діаграми діяльності забезпечують візуалізацію потоку керування програми на основі дій і діяльності. Діаграми станів застосовуються для відображення поведінки програмного засобу, використовуючи поняття станів об'єктів (систем) у певний момент часу та переходів між ними. Діаграми послідовності використовуються для подання взаємодії між об'єктами програми у часі. Діаграми комунікацій, на відміну від діаграм послідовності, більше призначені для відображення структурних, ніж часових, особливостей взаємодії між об'єктами. Діаграми взаємодії

відображають взаємодію між об'єктами у межах потоку керування. Діаграми синхронізації призначені для подання змін станів об'єктів та взаємодії між об'єктами у часі. Окрім того, для візуалізації динаміки виконання програмного коду можуть бути застосовані стандартні блок-схеми, діаграми потоків керування (CFD), діаграми потоків даних (DFD).

Існує значна кількість програмних засобів, що дозволяють побудувати блок-схему або UML-модель на основі програмного коду. Зокрема, вебзасіб CodeToFlow [3] дозволяє сформувати блок-схему, діаграму класів чи діаграму послідовності на основі програми, написаної однією з основних мов (C#, C, C++, Python, TypeScript, Java, JavaScript та ін.). Програмний засіб моделювання програмного забезпечення Visual Paradigm [4] дає можливість шляхом реверсної інженерії сформувати UML-діаграми класів, пакунків на основі коду, написаному на таких мовах, як Java, C++, C#, Python, Ada, CORBA IDL, PHP, Objective-C, XML. Доступна також генерація діаграм послідовності на основі Java-коду. Аналогічний засіб Enterprise Architect [5] забезпечує формування діаграм класів на основі мов Action Script, Ada, C, C#, C++, CORBA IDL, Delphi, Java, PHP, Python, SystemC, Verilog, VHDL, Visual Basic, Visual Basic .NET. Також варто згадати інструменти для автоматичного аналізу коду, які не лише будують UML-діаграми, а й виявляють потенційні проблеми у коді. Зокрема, SonarQube [6] надає детальну аналітику про стан коду та допомагає знайти вузькі місця у продуктивності або безпеці. Окрім відображення структури та особливостей поведінки програмного забезпечення, UML-моделі та блок-схеми можуть використовуватись для генерації коду. Якщо формування програмного коду здійснюється шляхом маніпуляції візуальними елементами, як зображення класів та зв'язків між ними, такий вид програмування називається візуальним [7]. Наприклад, Enterprise Architect [8] дозволяє згенерувати програмний код на основі діаграм класів, діяльності, станів, послідовності. Visual Paradigm [9] забезпечує формування коду на основі діаграм класів і станів. Іншим, більш спеціалізованим, прикладом візуального програмування може вважатись використання редактора вузлів у Blender [7], де замість прямого написання коду шейдерів застосовується комбінація шейдерних, геометричних, композитних, текстурних вузлів.

Окрім описаних стандартних напрямків візуалізації програмного забезпечення, варто зазначити сучасні тренди у візуалізації програмних структур. Одним із перспективних напрямків є застосування 3D-візуалізації коду, що дозволяє отримати об'ємне представлення архітектури проєкту та його компонентів. Такі засоби, як CodeCity [10] і Software Maps [11], дозволяють відобразити структуру програмного забезпечення у вигляді тривимірного міста, де будівлі відповідають класам, а вулиці — зв'язкам між ними. Ще одним цікавим напрямком є застосування графових баз даних для аналізу та візуалізації зв'язків між модулями коду. Наприклад, Neo4j [12] може бути використана для побудови графа залежностей класів, функцій або змінних у великому програмному проєкті.

Важливою є також візуалізація характеристик продуктивності програмного забезпечення при здійсненні профілювання (аналізу динамічних параметрів програми). Потужним засобом профілювання є Flame Graphs [13], що дозволяє зрозуміти, які складові програми займають найбільше часу для виконання. Фрагменти коду (функції) подаються у виді палючих прямокутників. Як правило, ширина прямокутника визначає час, витрачений на виконання відповідної функції. У результаті, легко виявити «гарячі» ділянки коду, які споживають найбільше часу чи ресурсів.

Для відображення динаміки розробки програмного забезпечення здійснюється візуалізація внесення змін (рис. 1) у репозиторій коду. Наприклад, програмний засіб Gource [14] дозволяє візуалізувати історію змін до репозиторію у формі анімації. Формується дерево, де коренем є центр репозиторію, гілками є папки, а листками — файли папок. При цьому, відображаються імена авторів внесених змін. У результаті, ріст дерева відображає розвиток проєкту.



Рис. 1. Приклад візуалізації історії розвитку репозиторію гри Minecraft [15]

Ще одним напрямом візуалізації є теплові карти коду, які дозволяють визначити найчастіше змінювані або найбільш проблемні частини коду. Інструменти, такі як CodeScene [16], аналізують історію комітів та виявляють "гарячі" зони проєкту.

Висновки

Візуалізація особливостей і характеристик програмного забезпечення сприяє ефективнішому аналізу його структури, динаміки та закономірностей розвитку. Використання UML і блок-схем допомагає глибше зрозуміти будову та поведінку програмної системи. Спеціалізовані програмні засоби дають змогу генерувати код на основі візуального опису або автоматично будувати схему програми на основі вихідного коду. Візуалізація результатів профілювання коду дозволяє наочно визначити найбільш трудомісткі фрагменти коду. Аналіз змін у репозиторії коду дозволяє виявити ключові етапи розвитку проєкту та їхній вплив на програмний продукт.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Романюк О. Н. Комп'ютерна графіка / О. Н. Романюк, О. В. Романюк, Р. Ю. Чехместрук. – Вінниця : ВНТУ, 2023. – 146 с.
2. What is Unified Modeling Language (UML)? [Online]. — Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> (Accessed on: 07.03.2025). — Title from screen.
3. Code to Flowchart: Visualize, Analyze, and Understand Your Code! [Online]. — Available: <https://codetoflow.com/> (Accessed on: 07.03.2025). — Title from screen.
4. Instant Reverse [Online]. — Available: https://www.visual-paradigm.com/support/documents/vpuserguide/276/277_instantrevert.html (Accessed on: 07.03.2025). — Title from screen.
5. Importing Source Code [Online]. — Available: https://sparxsystems.com/enterprise_architect_user_guide/17.0/modeling_domains/reverseengineersourcecode.html (Accessed on: 07.03.2025). — Title from screen.
6. SonarQube - Continuous Code Quality [Online]. — Available: <https://www.sonarqube.org/> (Accessed on: 07.03.2025). — Title from screen.
7. Завальнюк Є. К. Аналіз вимог до графічних мов програмування / Є. К. Завальнюк, О. Н. Романюк // Електронні інформаційні ресурси: створення, використання, доступ та управління. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 20-21 листопада 2024 р., [Суми/Вінниця], 20 — 21 листопада 2024 / НІКО / КЗВО «Вінницька академія безперервної освіти». — Суми/Вінниця, 2024. — С. 52—54.
8. Generate Source Code [Online]. — Available: https://sparxsystems.com/enterprise_architect_user_guide/17.0/modeling_domains/generatesourcecode.html (Accessed on: 07.03.2025). — Title from screen.
9. How to Generate Java from UML (Instant Generator Guide)? [Online]. — Available: https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/27971_instantgener.html (Accessed on: 07.03.2025). — Title from screen.
10. CodeCity – A Metaphor for Software Structure [Online]. — Available: <https://www.inf.usi.ch/phd/wettel/codecity.html> (Accessed on: 07.03.2025). — Title from screen.
11. Software Maps – Microsoft Research [Online]. — Available: <https://www.microsoft.com/en-us/research/project/code-maps/> (Accessed on: 07.03.2025). — Title from screen.
12. Neo4j Graph Database for Software Analytics [Online]. — Available: <https://neo4j.com/use-cases/software-analytics/> (Accessed on: 07.03.2025). — Title from screen.
13. CPU Flame Graphs [Online]. — Available: <https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html> (Accessed on: 07.03.2025). — Title from screen.
14. Gource [Online]. — Available: <https://github.com/acaudwell/Gource?tab=readme-ov-file> (Accessed on: 07.03.2025). — Title from screen.

15. 800+ days of Minecraft in 8 minutes [Online]. — Available: https://www.youtube.com/watch?feature=player_embedded&v=zRjTyRly5WA (Accessed on: 07.03.2025). — Title from screen.

16. CodeScene – Behavioral Code Analysis [Online]. — Available: <https://codescene.io/> (Accessed on: 07.03.2025). — Title from screen.

Завальнюк Євген Костянтинович — аспірант групи 121-23а, Вінницький національний технічний університет, Вінниця, e-mail: qq9272627@gmail.com

Складанюк Олексій Олегович — аспірант групи 121-23а, Вінницький національний технічний університет, Вінниця, e-mail: skladanyuk1999@gmail.com

Zavalniuk Yevhen K. — Postgraduate student, 121-23a group, Vinnytsia National Technical University, Vinnytsia, email : qq9272627@gmail.com

Skladaniuk Oleksiy O. — Postgraduate student, 121-23a group, Vinnytsia National Technical University, Vinnytsia, email : skladanyuk1999@gmail.com