Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://nv.nltu.edu.ua
https://doi.org/10.36930/40300514
Article received 03.11.2020 p.
Article accepted 19.10.2020 p.
UDC 004.514

ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

@ Correspondence author
R. T. Kovalchuk
roman.kovalchuk.lviv@gmail.com

Р. Т. Ковальчук, Т. О. Коротєєва
Національний університет "Львівська політехніка", м. Львів, Україна

АДАПТАЦІЯ ВЕРСІЇ АЛГОРИТМУ INTELLISENSE ДО ПРОФІЛЮ ПОТОЧНОГО ПРОЕКТУ

Проаналізовано декілька різних підходів до автодоповнення програмного коду та різні методи їхнього покращення. Досліджено системи автодоповнення програмного коду у різних середовищах розробки (Visual Studio, Visual Studio Code, Eclipse, IntelliJ, тощо) та для різних мов програмування (C#, Visual Basic, JavaScript, Java, Python, тощо). Детально проаналізовано алгоритм IntelliSense та його використання в середовищах розробки програмного забезпечення (ПЗ). Встановлено підходи, за допомогою яких розробники систем автодоповнення програмного коду досягали кращих результатів у знаходженні найкращої пропозиції стосовно завершення коду. Оглянуто системи, що використовують сторонні бази даних для кращого аналізу програмного коду користувача та знаходження найбільш відповідної пропозиції автодоповнення на підставі контексту. Розглянуто різні ідеї для кращого алгоритму та комбінацію декількох у певних системах. Встановлено, що жодна з наявних систем повністю не зменшує фізичну роботу розробника. Запропоновано нову ідею для покращення досвіду користувача під час написання програмного коду. Проведено модифікацію алгоритму IntelliSense, внаслідок чого він підтримує використання фільтрів проекту та впорядковує пропозиції відповідно до популярності їх використання у заданих фільтрах. Спроектовано базу даних, у якій зберігаються створені користувачем ПЗ профілі, їхній статус (увімкнутий чи вимкнений) та кількість використань різних доповнень у цих профілях. Розроблено програмний засіб, що використовує згадану вище модифікацію алгоритму IntelliSense із використанням мови програмування TypeScript та технології розроблення розширень до середовища розробки ПЗ Visual Studio Code. Інтегровано цей програмний засіб у середовище розробки Visual Studio Code за допомогою підтримуваних нею функцій. Проаналізовано приріст у зменшенні часу обирання варіанта доповнення програмного коду від параметрів використання цього програмного засобу.

Ключові слова: середовище розробки програмного забезпечення; автодоповнення програмного коду; Visual Studio Code; програмний проект.

Вступ

У сучасному світі з кожним днем створюється дедалі більше різного програмного забезпечення (ПЗ). Окрім цього, що росте потреба у новому ПЗ, зазвичай воно потрібне уже сьогодні. З цієї причини швидкість створення програм є дуже важливою як для його замовника, так і розробника. Існує багато різних факторів, які впливають на швидкість створення нового програмного продукту. Одним з таких факторів є швидкість написання програмного коду.

Написання програмного коду пришвидшують різними способами: застосуванням мови програмування більш високого рівня, підвищенням рівня кваліфікації розробників, інструментами, що допомагають створювати код, тощо. Обрати мову програмування вищого рівня буде не завжди правильно для проектів середнього та високого рівнів, оскільки зросте тривалість виконання функціоналу програми та її апаратні потреби. Підвищення рівня кваліфікації розробників, що працюють на проекті, є довготривалим і складним процесом, а пошук уже висококваліфікованих працівників часто не є легшим та дешевшим процесом.

Одним з інструментів, що дещо пришвидшує розроблення ПЗ, є автодоповнення програмного коду. У сучасних середовищах розробки уже інтегроване автодоповнення програмного коду. Також у популярних середовищах розробки ПЗ існують розширення, що покращують стандартні вбудовані автодоповнювачі коду.

Однак наявні системи автодоповнення програмного коду містять недоліки та не завжди пропонують доповнити код саме так, як потрібно. Отже, дослідження у цій сфері є необхідними й, що найважливіше, актуальними.

Об'єкт дослідження – автодоповнення програмного коду у середовищах розробки ПЗ.

Предмет дослідження – модифікація алгоритму автодоповнення програмного коду IntelliSense так, щоб він підтримував застосування фільтрів проекту та впорядковував пропозиції відповідно до популярності їх використання у заданих фільтрах.

Мета роботи – домогтися пришвидшення процедури написання програмного коду способом автодоповнення текстом, введеним користувачем в інтегроване середовище розробки ПЗ, адаптованим як до нього, так і до поточного програмного проекту.

Для досягнення зазначеної мети визначено такі *основні завдання дослідження:*

Інформація про авторів:
Ковальчук Роман Тарасович, магістрант, кафедра програмного забезпечення. **Email:** roman.kovalchuk.lviv@gmail.com
Коротєєва Тетяна Олександрівна, канд. техн. наук, доцент, кафедра програмного забезпечення.
Email: tetyana.o.korotyeyeva@lpnu.ua

Цитування за ДСТУ: Ковальчук Р. Т., Коротєєва Т. О. Адаптація версії алгоритму IntelliSense до профілю поточного проекту. Науковий вісник НЛТУ України. 2020, т. 30, № 5. С. 84–89.
Citation APA: Kovalchuk, R. T., & Korotyeyeva, T. O. (2020). Adaptation of the IntelliSense algorithm to the profile for per project usage. *Scientific Bulletin of UNFU*, *30*(5), 84–89. https://doi.org/10.36930/40300514

1. Дослідити програмну систему, яку необхідно розширити.
2. Розробити алгоритм системи автодоповнення програмного коду.
3. Розробити архітектуру бази даних, необхідної для роботи програми.
4. Інтегрувати розширення у середовище розробки програмного коду Visual Studio Code.
5. Дослідити час на написання програмного коду з використанням створеної системи та без неї.

Наукова новизна отриманих результатів дослідження – розроблено модифікофаний алгоритм автодоповнення програмного коду, що використовує профіліфільтри проекту для впорядкування пропозицій доповнення та базується на наявній системі IntelliSense.

Практична значущість результатів дослідження – запропонований алгоритм дасть змогу зменшити час на написання програмного коду за рахунок застосування фільтрів проекту та впорядкування можливих пропозиції відповідно до їх популярності використання у заданих фільтрах.

Аналіз актуальних досліджень і публікацій. Оскільки проблема у швидкості написання програмного коду є актуальною, то існує багато продуктів, що її вирішують як на апаратному, так і програмному рівнях. Одним з способів пришвидшення процедури написання програмного коду є його автодоповнення відповідним текстом. Найпопулярніші середовища розробки ПЗ містять вбудовані компоненти, що доповнюють код.

Програмісти використовують автодоповнення для зменшення когнітивних витрат на запам'ятовування вичерпних списків API впродовж багатьох років. Автодоповнення має первинний і очевидний момент відмови: коли програміст очікує існування певного методу чи назви функції, а це не відбувається, тому перелік автодоповнення просто припиняє відображати результати та зникає. Раніше було описано автоматичне визначення функції (AFD) [4], яке може досягти успіху, коли не завершено автоматичне завершення. Це спосіб зменшити вплив бібліотек, що не працюють у потоці, збільшити швидкість кодування основних завдань програмування та розподілити роботу між різними типами програмістів і автоматичних інструментів. Замість того, щоб побачити порожній перелік, користувачі можуть замість цього зробити автоматичне визначення функції, що використовує декілька джерел для визначення функції, яку користувач мав намір використовувати. Автори представили три допоміжні джерела визначення функцій на підставі інформації про функцію, яку користувач надає під час написання коду: пошук відповідного коду, колеги-програмісти та інші.

У роботі [7] зазначено, що автодоповнення програмного коду – це широко використовуваний інструмент для покращення продуктивності розробника ПЗ. Воно знімає потребу у програміста запам'ятовувати та вводити точні назви методів або класів. Автори також згадали, що попри те, що автодоповнення програмного коду використовується дуже часто, однак існує мало його удосконалень. Виділено причини, через які автодоповнення програмного коду не розвивалось:

- не було очевидного способу покращити процедуру автодоповнення, що залежить від вибраної мови програмування. Наявні алгоритми вже враховували структуру програми та структуру API. Щоб покращити їх, потрібні були додаткові джерела інформації;
- немає чіткого порівняння між алгоритмами, щоб показати, що один з них кращий за інші;
- прості алгоритми автодоповнення програмного коду вже давали багато користі, тому їхнє покращення не було б відчутно помітним.

Алгоритми автодоповнення програмного коду використовують великий обсяг інформації для того, щоб зменшити кількість пропозицій. Для прикладу, програмний код, написаний мовою програмування Java, пропонуватиме використати тільки методи класу String, якщо звертаються до об'єкта типу String.

Автори [7] вважають попередні алгоритми "песимістичними", позаяк вони повертають багато варіантів автодоповнення, посортованих в алфавітному порядку. Незважаючи на простоту реалізації, такий метод швидко виконується, що є його позитивним моментом. На противагу попередньо згаданим алгоритмам, автори [7] представили "оптимістичний" алгоритм автодоповнення. Його положення:

- кількість збігів до кожного доповнення є обмеженою (в їхньому варіанті кількість збігів обмежена до трьох);
- варіант, який необхідний програмісту, з високою ймовірністю опиниться в переліку повернутим алгоритмом;
- щоб мінімізувати кількість введеного тексту, префікс доповнення повинен бути коротким.

Одним з успішно реалізованих прикладів автодоповнення програмного коду є "Calcite" [3] – плагін до середовища розробки Eclipse, який допомагає вирішити труднощі з розумінням і правильним використанням API. Цей плагін рекомендує використати клас чи функцію, опираючись на приклади програмного коду, що були написані раніше. Для того, щоб швидко додати доповнений код до наявного, плагін Calcite додає пропозиції до переліку можливих варіантів у спливаючому вікні. Обрана пропозиція доповнить наявний програмний код. Цей плагін також використовує код, написаний іншими програмістами, щоб зробити кращу пропозицію доповнення. У ході дослідження коду, написаного різними користувачами, плагін Calcite покращив успішність пропозиції доповнення на 40 %.

Основні особливості плагіна Calcite:

- доповнення до меню завершення – це відображення додаткової інформації щодо методу, який обирається;
- вставка в код користувача. Після того, як користувач обирає потрібний метод, плагін Calcite вставляє код в інші лінійки. Наприклад, бувають випадки, коли необхідно імпортувати додаткові бібліотеки, тоді плагін Calcite виконує це за користувача;
- меню автодоповнення не вмикається автоматично у разі введення символу '='. Ця особливість плагіну Calcite мотивується тим, що не відомо, потрібно створити новий об'єкт класу чи присвоїти змінній уже наявне значення. Автодоповнення можна використати за допомогою комбінації клавіш control-space.

Розробники [3] створили базу даних, що зберігає найчастіше використані конструкції автодоповнення програмного коду. Одним із джерел заповнення цієї бази даних є відомий пошуковик Yahoo.

У роботі [8] автори навели результати дослідження, яке оцінює механізми автодоповнення програмного коду популярних середовищ розробки Java та.Net IDE. Це показує, що між ними є істотні відмінності, і що з хорошим механізмом заповнення коду програмісти частіше знаходять найоптимальніші методи та перевантаження, зменшуючи складність отриманого коду та покращуючи загальну продуктивність їхньої роботи.

Автори [8] вважають найважливішою відмінністю у проаналізованих механізмах те, як відображаються методи на перевантаження. Перший тип рішення використовується у середовищах розробки Eclipse, NetBeans та IntelliJ. У ньому інформація подається в одному вікні. Такі середовища, як Visual Studio, SharpDevelop та MonoDevelop використовують два кроки: спочатку користувач обирає метод, а потім конкретне його перевантаження. Система ReSharper також показує два кроки, але одночасно у двох вікнах.

Хоча завершення програмного коду неминуче для ефективного його редагування в інтегрованих середовищах розробки, наявні інструменти для завершення коду все ж таки можна вдосконалити. Наприклад, у роботі [5] зазначено: "Нещодавно введеному кандидату для заповнення коду варто присвоїти більш високий рейтинг серед раніше вставлених елементів у переліку кандидатів". Для підтвердження цього твердження в цьому документі розглядають такі три моменти. По-перше, експеримент із використанням історій операцій подано так, щоб підтвердити, що розробники найчастіше повторюють останні операції зі завершення коду. По-друге, наведено приклад роботи інструменту RCC Candidate Sorter. І останнє, наводять результати експерименту для оцінювання роботи цього інструменту. Отже, розроблена авторами система запам'ятовує три типи операцій:

- *звичайні операції* – до яких автори відносять видалення, вставлення чи модифікацію тексту. Ці операції виконуються вручну програмістом;
- *операції з'єднання* – виконуються автоматично середовищем розробки;
- *операції меню* – виконуються за допомогою функцій, прописаних у меню. Також до цих операцій належать ті, що виконуються за допомогою комбінації клавіш.

Уся історія операцій поділена на сесії розроблення ПЗ. Припускається, що розробник не може бути уважним після довгих перерв, тому сесія завершується після тридцяти хвилин без операцій.

Для нормалізації введеного тексту автори [5] пропонують не враховувати такі елементи: операції, що були відмінені; символи пунктуації; тип параметрів; символи, що перешкоджають канонічному порівнянню.

Меню доповнення програмного коду використовують більшість розробників, оскільки вони тісніше інтегровані в робочий процес розроблення ПЗ. Покращення меню доповнення коду, що містять додаткові джерела інформації, виявилося вельми потрібним завданням.

Іншим інструментом для автодоповнення програмного коду є технологія IntelliSense [2] – це утиліта середовища розробки Visual Studio [9]. Функції утиліти допомагають отримати більше інформації про програмний код, що використовується, відстежувати введені параметри та додавати виклики до властивостей та методів тільки кількома натисканнями клавіш. Багато особливостей технології IntelliSense залежать від мови програмування.

Visual Studio IntelliCode [10] – це версія алгоритму IntelliSense, що використовується у середовищі розробки Visual Studio та використовує штучний інтелект задля кращих пропозицій доповнити програмний код. Алгоритм IntelliCode розширює наявні робочі процеси для розробників за допомогою сервісів машинного навчання, які забезпечують розуміння програмного коду та його контексту. Вже широко застосовується для мови програмування C#, C++, JavaScript/TypeScript та XAML та оновлюватиметься в майбутньому для підтримки більшої кількості мов.

Visual Studio Code [1] – це легкий, але потужний редактор програмного коду, який доступний для Windows, macOS та Linux. Він оснащений вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов програмування (таких як C++, C#, Java, Python, PHP, Go) та середовищ виконання (таких як.NET і Unity). У цьому середовищі користувач може параметризувати алгоритм Intellisense.

Для розроблення систем автодоповнення програмного коду часто використовують машинне навчання. Автори [6] демонструють вирішення проблеми синтезу автодоповнення коду. Отримавши програмний код, вони синтезують місця, куди потрібно вставити доповнення. Основна ідея полягає у тому, щоб зменшити проблему автодоповнення програмного коду за допомогою використання машинного навчання та ймовірностей появи необхідного доповнення. Проведено простий і масштабований статичний аналіз, який витягує послідовності викликів методів з великої бази даних з програмним кодом й індексує їх у статистичну модель мови програмування. Після цього використовується модель мови, щоб знайти речення з найвищим рейтингом і використати у синтезі автодоповнення. Цей підхід здатний синтезувати послідовності викликів за декількома об'єктами разом з їх аргументами. Практично всі обчислені завершення набору доповнень, і бажане завершення відображається в топ-3 результатів у 90 % випадків.

Результати дослідження та їх обговорення

Алгоритм автодоповнення програмного коду із використанням профілю проекту. Більшість програмістів розробляють різні типи проектів. Кожен з проектів потребує різних доповнень при різних чи однакових контекстах. Для покращення досвіду користувача було розроблено систему, що надає варіанти продовження згідно з фільтром, що увімкнув користувач, який визначає тип проекту. Ці фільтри називають профілями.

Складні проекти часто бувають різних типів, отже – ця система дає змогу використовувати різні типи проектів (профілі). Користувач має змогу змінювати профілі під час написання програмного коду.

Алгоритм системи IntelliSense працює так:

1. Спрацьовує на комбінацію клавіш Ctrl+Space або після введення символу ".".
2. Зчитує введений текст від початку команди до позиції, в якій було викликано автодоповнення.
3. Знаходить команди, що містять введений текст, за допомогою мовного сервісу.
4. Знаходить усі члени класу, до об'єкта якого звертається користувач, за допомогою рефлексії та обирає тільки ті, що містять уже введений текст.
5. Впорядковує команди та знайдені доступні члени класу раніше, надаючи пріоритет частіше введеним раніше пропозиціям, використовуючи ранг пропозиції.
6. Виводить впорядковані пропозиції у новому вікні середовища розробки.
7. При виборі пропозиції доповнює програмний код нею та закриває вікно з пропозиціями.

Ранг пропозиції обчислюється за такою формулою:

$$Rank = f(x),$$

де: $Rank$ – ранг пропозиції; $f(x)$ – функція, яка визначає кількість використань пропозиції у системі.

Під час дослідження було розроблено алгоритм автодоповнення програмного коду із використанням профілю проекту. Після кожного вибору тексту автодоповнення у середовищі розробки ПЗ розроблений алгоритм накопичує кількість використань цього автодоповнення при увімкнутих профілях у цей момент.

Дані про використання зберігаються у базі даних, що підтримується середовищем розробки. На рис. 1 зображено діаграму цієї бази даних.

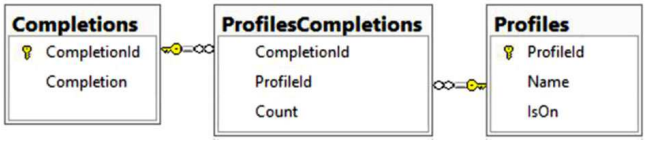

Рис. 1. Діаграма бази даних для роботи алгоритму

У таблиці Profiles зберігаються усі наявні фільтри проектів – профілі. Ця таблиця містить ідентифікатор профілю, його назву та значення чи він зараз увімкнутий. У таблиці Completions зберігаються усі використані автодоповнення. Ця таблиця містить ідентифікатор автодоповнення та сам текст автодоповнення.

У таблиці ProfilesCompletions зберігаються дані про використані автодоповнення при увімкнутих певних профілях. Ця таблиця містить ідентифікатори профілю та автодоповнення, а також кількість використань.

Під час кожного вибору автодоповнення розроблена система оновлює поле Count у таблиці ProfilesCompletions у рядках, у яких ідентифікатор автодоповнення позначає обране автодоповнення та ідентифікатор профілю позначає профіль, який зараз увімкнутий. У разі відсутності рядка з такими ідентифікаторами система створює рядок зі значенням "1" у полі Count.

Під час кожного виклику вікна автодоповнення система перевіряє дані про кожне можливе автодоповнення. Для цього використовується такий SQL скрипт, у якому параметр @completions – це перелік можливих автодоповнень:

```
SELECT c. Completion, SUM (pc.Count) as Count
FROM ProfilesCompletions pc
JOIN Profiles p ON p. ProfileId = pc.ProfileId
JOIN Completions c ON c. CompletionId = pc.CompletionId
WHERE p. IsOn = 1 AND c. Completion IN @completions
GROUP BY c. Completion.
```

Після отримання даних система відсортовує пропозиції у порядку спадання кількості їхніх використань, отриманих з бази даних. Відповідно, пропоноване автодоповнення, яке матиме найбільшу кількість використань, пропонується першим у переліку пропозицій. Отже, першою у переліку пропозицій буде пропозиція із найбільшим рангом. Цей ранг обчислюється за формулою:

$$Rank = \sum_{i=1}^{n} f(x_i),$$

де: $Rank$ – ранг пропозиції; $f(x_i)$ – функція, яка визначає кількість використань у конкретному профілі.

Систему автодоповнення програмного коду із використанням профілю проекту було інтегровано у середовище розробки ПЗ Visual Studio Code, яку створено з урахуванням можливості розширення. Від інтерфейсу користувача до редагування майже кожну частину VS Code можна налаштувати та вдосконалити за допомогою API розширення. Багато основних функцій VS Code побудовані як розширення та використовують той самий API розширення, який пропонується іншим розробникам.

Щоб побудувати розширення, було використано:

1. Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript.
2. Git – розподілена система керування версіями файлів і спільної роботи.
3. Yeoman – Yeoman допомагає запускати нові проекти, прописуючи найкращі практики та інструменти, які допоможуть залишатись продуктивним.
4. Пакет generator-code за допомогою команди "npm install -g yo generator-code".

Створене розширення виконує такі дії:

1. Реєструє подію активації onCommand: onCommand: extension.projectName, це дає можливість розширенню активуватись, коли користувач запускає команду з назвою розширення.
2. Використовує contributes.commands, щоб зробити команду розширення доступною на панелі команд та прив'язати її до розширення з ідентифікатором команди.
3. Використовує API commands.registerCommand для прив'язки функції до зареєстрованого розширення з ідентифікатором команди.

Розроблене розширення до середовища розробки VS Code містить package.json як маніфест розширення. Package.json містить поєднання полів Node.js, таких як *scripts* та *dependencies*, та специфічних полів VS Code, таких як *publisher*, *activationEvents* та *contributes*. Розширення було розроблено за допомогою мови програмування TypeScript, що розширює можливості JavaScript.

Під час першого використання було створено новий профіль, внесено його та викликано вікно автодоповнення за допомогою комбінації клавіш Ctrl+Space (рис. 2) та обрано другу пропозицію з переліку (react-bootstrap). Після наступного виклику вікна автодоповнення (рис. 3) пріоритетною пропозицією була "react-bootstrap", оскільки вона вже використовувалась у цьому профілі.

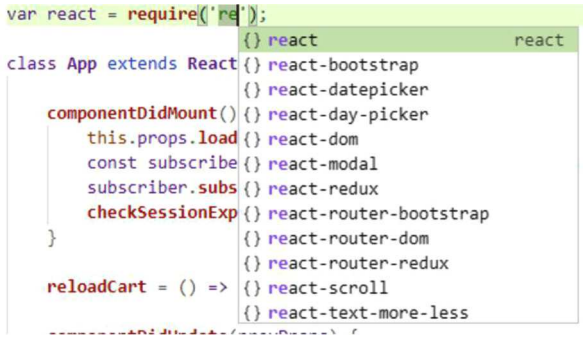

Рис. 2. Перелік пропозицій під час першого виклику вікна доповнень

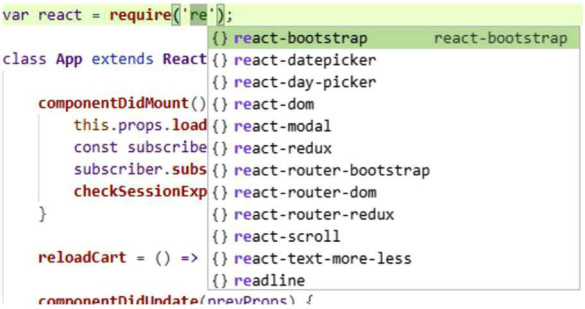

Рис. 3. Перелік пропозицій під час другого виклику вікна доповнень

Проведено дослідження, під час якого визначено приріст у швидкості розробки, залежно від того, використовувались профілі чи ні.

Спочатку без використання було використано 100 автодоповнень програмного коду з таблиці. Перших 50 використано з одної групи доповнень, решта – з іншої. Під час використання другої половини доповнень користувачу пропонувались пріоритетними доповнення з першої групи. Місце потрібних доповнень у переліку для другої половини доповнень було з 1 по 4. На рис. 4 зображено залежність порядку необхідного доповнення від номера доповнення для цього експерименту з використанням алгоритму IntelliSense. Для вибору необхідного доповнення було виконано 179 дій.

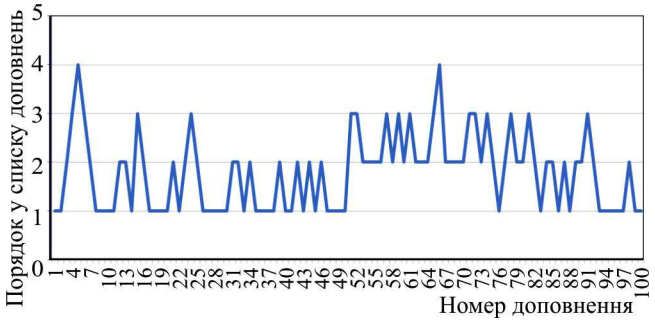

Рис. 4. Порядок необхідного доповнення у списку всіх доповнень з використанням алгоритму IntelliSense

Для використання системи з профілями було створено два профілі та названо їх "перший" та "другий". Увімкнуто перший профіль та використано ті ж самі 50 доповнень, що використовувались раніше. Після цього перший профіль було вимкнено та увімкнуто другий. Використано 50 наступних автодоповнень. Автодоповнення були пріоритетними, якщо вони раніше використовувались в другому профілі. Більшість автодоповнень з другої половини були першими у переліку запропонованих. На рис. 5 зображено залежність порядку необхідного доповнення від номера доповнення для даного експерименту з використанням модифікованого алгоритму. Для вибору необхідного доповнення було виконано 146 дій.

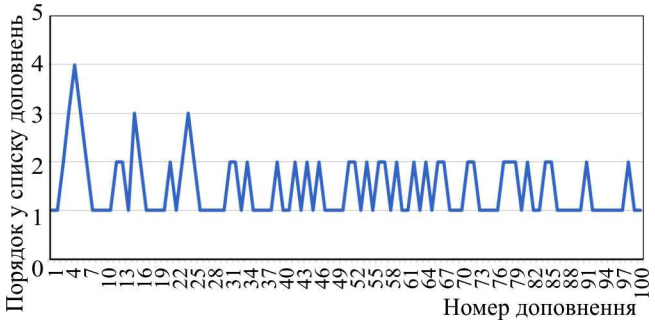

Рис. 5. Порядок необхідного доповнення у списку всіх доповнень з використанням модифікованого алгоритму

Порівнюючи розроблену систему автодоповнення програмного коду із використанням профілю проекту та систему IntelliSense, було визначено, що для першої половини доповнень з таблиці дві системи надали однакові результати, але для другої половини доповнень розроблена система надала значно кращі результати: необхідне доповнення завжди було впорядковане на першому чи другому місці у списку, тоді як доповнення у системі IntelliSense було часто на третьому місці. Використання створеного алгоритму дозволило використати на 32 дії менше порівняно з класичним алгоритмом IntelliSense. За допомогою системи автодоповнення програмного коду із використанням профілю проекту було виконано на 17,9 % дій менше, ніж потрібно для отримання аналогічного результату, використовуючи алгоритм IntelliSense.

Таблиця. Використані доповнення під час тестування

Введений текст	Очікуване доповнення	Група доповнень	Кількість використань
re	react	Перша	10
co	component	Перша	5
PackageCo	PackageCompletionItem	Перша	5
make	makeStyles	Перша	3
ma	map	Перша	2
fi	filter	Перша	5
clas	classes	Перша	6
set	setPosition	Перша	4
get	getPosition	Перша	7
cre	create	Перша	3
re	redux	Друга	5
co	componentDidMount	Друга	8
PackageCo	PackageCompletions	Друга	2
make	makeVisible	Друга	7
ma	mapper	Друга	3
fi	find	Друга	5
clas	className	Друга	4
set	setState	Друга	4
get	getState	Друга	6
cre	createRequest	Друга	6

За допомогою цього експерименту було підтверджено, що використання профілів як фільтрів процедури проектування зменшило кількість дій для вибору правильного автодоповнення.

Висновок

Проаналізовано системи автодоповнення програмного коду та їхнє використання у середовищах розробки ПЗ. Спроектовано модифікацію алгоритму автодоповнення за допомогою використання профілів (фільтрів) проекту. Розроблено цю модифікацію за допомогою мови програмування TypeScript та інтегровано її в алгоритм IntelliSense. Розгорнуто цей програмний засіб у середовищі розробки ПЗ Visual Studio Code.

Проведено дослідження, за допомогою якого встановлено, що розроблений програмний засіб надає користувачу потрібні доповнення програмного коду частіше, ніж без його використання, отже – пришвидшує написання коду за допомогою зменшення часу на обирання потрібної пропозиції автодоповнення. За допомогою розробленої системи автодоповнення програмного коду із використанням профілю проекту користувачем було виконано на 17,9 % дій менше, ніж потрібно для отримання аналогічного результату, використовуючи алгоритм IntelliSense. Встановлено, що цей програмний засіб актуальний тільки для користувачів, що використовують різні стилі написання програмного коду та різні типи проектів.

Розроблену модифікацію алгоритму автодоповнення можливо адаптувати до інших програмних засобів й інтегрувати в різні середовища розробки ПЗ.

References

1. Documentation for Visual Studio Code. (2020). *Visual Studio Code.* Retrieved from: https://code.visualstudio.com/docs

2. IntelliSense in Visual Studio. (2020). *Microsoft Docs*. Retrieved from: https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019.
3. Mooty, M., Faulring, A., Stylos, J., & Myers, B. (2010). Calcite: Completing Code Completion for Constructors Using Crowds. *IEEE Symposium on Visual Languages and Human-Centric Computing*. https://doi.org/10.1109/VLHCC.2010.12
4. Murray, K., & Bigham, J. (2011). Beyond Autocomplete: Automatic Function Definition. *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing*. https://doi.org/10.1109/VLHCC.2011.6070421
5. Omori, T., Kuwabara, H., & Maruyama, K. (2015). Improving code completion based on repetitive code completion operations. *Information and Media Technologies, 10*, 210–225. https://doi.org/10.11185/imt.10.210
6. Raychev, V., Vechev, M., & Yahav, E. (2014). Code Completion with Statistical Language Models. *ACM SIGPLAN Notices*. https://doi.org/10.1145/2666356.2594321
7. Robbes, R., & Lanz, M. (2008). *How Program History Can Improve Code Completion*. DBLP. https://doi.org/10.1109/ASE.2008.42
8. Scheller, T., & Kuehn, E. (2013). Influence of Code Completion Methods on the Usability of APIs. *Proceedings of the IASTED International Conference Software Engineering* (SE'2013). https://doi.org/10.2316/P.2013.796–027
9. Visual Studio documentation. (2019). *Microsoft Docs*. Retrieved from: https://docs.microsoft.com/en-us/visualstudio/?view=vs-2019
10. Visual Studio IntelliCode. (2020). *Visual Studio Marketplace*. Retrieved from: https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.VSIntelliCode.

R. T. Kovalchuk, T. O. Korotyeyeva
Lviv Polytechnic National University, Lviv, Ukraine

ADAPTATION OF THE INTELLISENSE ALGORITHM TO THE PROFILE FOR PER PROJECT USAGE

Several different code autocompletion systems and different methods of their improvement are analyzed. The systems of autocompletion of the source code in different integrated development environments (Visual Studio, Visual Studio Code, Eclipse, IntelliJ, etc.) and for different programming languages (C #, Visual Basic, JavaScript, Java, Python, etc.) are studied. The IntelliSense algorithm and its use in software development environments are analyzed in detail. Approaches have been established by which the developers of the source code autocompletion systems have achieved the best results in finding the best proposal for the code suggestions. Systems that use third-party databases are reviewed to better analyze user code and find the most appropriate context-based autocomplete suggestions. Different ideas for a better algorithm and a combination of several algorithms in certain systems are considered. It is revealed that none of the existing systems completely reduces the physical work of the developer. A new idea has been developed to improve the user experience when writing program code. The IntelliSense algorithm has been modified to support the use of project filters and to arrange the proposals according to the frequency of the use of these suggestions in the specified filters. A database has been designed to store user-created profiles, their status (on or off) and the number of usages of various add-ons in these profiles. A software tool was developed that uses the aforementioned modification of the IntelliSense algorithm using the TypeScript programming language and technology to develop extensions to the Visual Studio Code software development environment. This software was integrated into the development environment using supported Visual Studio Code features. The reduction of the time for choosing the option to complete the program code from the use of this software is analyzed.

Keywords: integrated development environment; source code autocompletion; Visual Studio Code; software project.