УДК 004.4'22 https://doi.org/10.15407/pp2021.01.026

А.М. Покровський

ЗАСІБ ВИМІРЮВАННЯ МЕТРИК ВИХІДНОГО КОДУ FORTRAN ЗА ДОПОМОГОЮ СИНТАКСИЧНОГО АНАЛІЗУ

За умов стрімкого розвитку методик забезпечення якості програмного коду, все більшою стає потреба в інструментах, що можуть автоматизувати процес оновлення та реструктуризації текстів програм. Розроблено програмний засіб для вимірювання програмних метрик, що дозволяє провести оцінювання якості вихідного коду програм мовою Fortran. Для цього розроблено алгоритми обходу синтаксичного дерева програми та на їх основі реалізовано модуль для інтегрованого середовища програмування Photran. Модуль використовує синтаксичний аналізатор програмного коду та побудоване Photran на його основі структурне дерево. Проведено порівняння розробленого засобу з наявними інструментами аналізу вихідного коду.

Ключові слова: метрика вихідного коду, якість програмного забезпечення, рефакторинг.

Вступ

Забезпечення якості вихідного коду – це важлива проблема, що стоїть перед розробниками програмного забезпечення. Протягом життєвого циклу програмного продукту вимоги до нього видозмінюються, що призводить до необхідності редагування та реструктуризації коду. У деяких випадках, життєвий цикл програми (від початку проектування до виходу із вжитку) може складати десятки років. За такий час міняються не тільки функціональні вимоги, а й версії мови програмування, що використовується, стилістичні підходи при написанні текстів програм тощо.

Прикладом такого середовища є екосистема мови Fortran. Як перша високорівнева мова програмування, що активно використовується й нині, Fortran за час свого існування зазнала значних змін як у синтаксисі, так і у вимогах якості до вихідного коду [1]. Водночас, велика кількість програмних засобів та компонент вимагає підтримки та оновлення, оскільки задачі, що вирішуються за допомогою цих програм, не втрачають своєї актуальності. Для забезпечення підтримуваності та для спрощення задачі проектування Fortran – систем була створена значна кількість підручних засобів автоматизації розробки (Computer-Aided Software Engineering, CASE-інструментів). Процес підвищення якості програмного забезпечення (ПЗ) (реінжиніринг) полягає у виявленні та усуненні потенційних вад (технічних або стилістичних), що можуть ускладнити роботу над програмним продуктом. Для цього CASE-інструменти надають дві групи автоматизованих операцій – вимірювання програмних метрик, що дозволяють дати оцінку якості вихідного коду, та здійснення рефакторингів, тобто внесення у цей код комплексних змін, що призведуть до підвищення його якості. Існує багато інтегрованих середовищ програмування, які надають подібні можливості при роботі з Fortran, наприклад, Forcheck, fpt, плагіни для Visual Studio та CLion.

В даній роботі за основу програмного засобу було взято Photran – плагін для Eclipse IDE з відкритим вихідним кодом, що містить вбудований програмний інтерфейс для взаємодії та аналізу вихідного коду Fortran-програм. На його основі створено засіб для оцінювання якості вихідного коду за допомогою вимірювання програмних метрик. Також наведено порівняння різних інструментів, що надають подібний функціонал.

Матеріал даної роботи організовано наступним чином. В першому розділі описані програмні метрики, вимірювання яких було реалізовано у розробленому засобі. В другому розділі описано плагін Photran та реалізація інтерфейсу програмного засобу. В третьому розділі описано розроблені алгоритми вимірювання метрик з використанням синтаксичного дерева програми. В останньому розділі наведено порівняння розробленого програмного засобу зі схожими інструментами, що надають інші середовища програмування.

1. Програмні метрики

Метрики програмного забезпечення – множина алгоритмів, за допомогою яких можна отримати з вихідного коду програми числове значення, яке має свідчити про рівень складності, взаємозалежності та ефективності алгоритму. Більшість метрик фокусуються на вимірюванні таких показників, як розмір програми, складність логічного графу програми (структурні елементи та залежності між ними – класи, модулі, файли, функції тощо) або складність графу керування програми (послідовність виконання інструкцій, викликів підпрограм).

Різні класи метрик вимагають для їх обчислення взаємодії з різними форматами представлення програми. Для максимально повної демонстрації можливостей програмного інтерфейсу засобу доцільно вибирати метрики, які покривають більшість таких форматів.

В роботі реалізовано вимірювання трьох метрик, що представляють три класи взаємодій з програмним інтерфейсом.

Кількість рядків коду (Source Lines Of Code, SLOC) – примітивна метрика, що вимірює обсяг та зусилля, необхідні для написання програми рядками вихідного коду. Ця метрика складається з трьох різних значень:

- кількість фізичних рядків коду (LOC) – кількість рядків програми, що безпосередньо перетворюються компілятором/транслятором на машинний код;
- кількість рядків коментарів (comment LOC, CLOC) – кількість рядків коментарів, тобто тверджень, що ігноруються компілятором;
- кількість пустих рядків (blank LOC, BLOC) – кількість рядків, що не містять тексту та використовуються для візуального розбиття програми на блоки.

Вид взаємодії з програмним інтерфейсом – для вимірювання цієї метрики необхідно мати змогу отримати доступ до вихідного коду програми.

Обсяг зусиль по Холстеду (Halstead Effort) – метрика, що була введена в 1977 році Морісом Холстедом в числі групи метрик складності (Halstead Complexity Measures) як спосіб виміряти складність та обсяг вихідного коду програми. Ця метрика фокусується на оцінюванні зусиль, що будуть витрачені програмістом на написання або підтримку коду та розглядає текст програми як набір лексичних одиниць та зв'язків між ними [2].

Для обчислення обсягу зусиль за Холстедом необхідно виміряти такі початкові значення:

n_1 – кількість унікальних операторів,

n_2 – кількість унікальних операндів,

N_1 – загальна кількість операторів,

N_2 – загальна кількість операндів.

Ці значення характеризують розмір та різноманітність програмного лексикону, що був використаний при написанні вихідного коду, а відношення між загальною кількістю та кількістю унікальних елементів показує ступінь зв'язаності тексту (більша кількість використань символу – більша зв'язаність). На основі цих значень здійснюється обчислення похідних метрик:

- словник програми (n):

$$n = n_1 + n_2 . \quad (1)$$

- розмір програми (N):

$$N = N_1 + N_2 . \quad (2)$$

- обсяг (об'єм) програми (V):

$$V = N \times \log_2 n . \quad (3)$$

- складність програми (D):

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} . \quad (4)$$

- обсяг зусиль для написання програми (E):

$$E = D \times V . \quad (5)$$

Вимірювання цієї метрики на відміну від LOC вимагає аналізу вихідного коду та визначення залежностей між лексичними одиницями мови програмування. Вид взаємодії з програмним інтерфейсом – збір інформації про синтаксичні елементи ви-

хідного коду, їх тип та зв'язки між визначенням та використанням ідентифікаторів.

Цикломатична складність за Маккейбом – структурна метрика складності алгоритму, введена Томасом Маккейбом в 1976 році. Практичною проблемою, що призвела до створення цієї метрики, була необхідність вирахування кількості тестів за допомогою яких можна повністю покрити комп'ютерну програму. Цикломатична складність визначається як кількість лінійно незалежних шляхів через алгоритм [3]. Найпростіша програма, що не містить операторів умовного переходу, буде мати цикломатичну складність 1, бо існує лише один шлях проходу через неї. Один умовний перехід збільшить кількість маршрутів удвічі – один у випадку коли значення умови переходу є істинним, другий у випадку коли воно є хибним.

З математичної точки зору цикломатична складність вираховується за допомогою представлення програми у вигляді орієнтованого графа, де вузлами є групи нероздільних інструкцій, а ребрами – переходи потоку керування між цими групами. Позначимо:

E – кількість ребер у графі,

N – кількість вузлів у графі,

P – кількість компонентів зв'язності графа (для підпрограми дорівнює 1).

Тоді цикломатичну складність M можна обчислити за формулою:

$$M = E - N + 2 \times P .$$

Для вимірювання цієї метрики необхідно побудувати та обійти структурний граф програми та визначити, як впливають на цикломатичну складність різні мовні конструкції. Вид взаємодії з програмним інтерфейсом – збір інформації про потік керування та послідовність виконання інструкцій програми.

2. Середовище Photran

Photran – потужне інтегроване середовище програмування (IDE) для Fortran побудоване на базі Eclipse. В набір функцій Photran входять редагування вихідного коду, система пошуку та навігації, компіляція, виконання та налагодження програм. Для побудови проектів використовується make, що в сукупності з відкритим вихідним кодом та можливістю запускати Eclipse з усіх популярних ОС, робить Photran зручним та гнучким засобом, який можна швидко налаштувати під конкретні потреби.

Представлення Fortran-програми у Photran здійснюється за допомогою двох структур даних – абстрактного синтаксичного дерева (Abstract Syntax Tree, AST) та віртуальнного програмного графа (Virtual Program Graph, VPG). AST представляє собою структурне дерево програми, гілки якого закінчуються листками, які відповідають за синтаксичні елементи мови програмування (оператори, літерали, ідентифікатори тощо). VPG служить інтерфейсом між програмістом та AST, дозволяє отримувати та вивільнювати його екземпляри, проводити лексичний аналіз коду, отримувати інформацію про зв'язки між програмними структурами та їх ідентифікаторами [4].

Модуль вимірювання метрик необхідно пов'язати з інтерфейсом Eclipse для можливості взаємодії з вікном огляду проекту (для вибору файлів, з яких необхідно зібрати дані). Для цього у роботі було використано вбудовану у Photran механіку рефакторингів. Вона дозволяє за допомогою так званого «ресурсного рефакторингу» зареєструвати певну операцію, що має відповідний підпункт на навігаційній панелі та передає на вхід операції файли, що були виділені користувачем у вікні огляду проекту.

Для реєстрації рефакторингу необхідно виконати ряд дій:

- створити клас, що реалізує інтерфейс IResourceRefactoring. Цей інтерфейс містить методи для отримання назви операції, перевірки початкових умов (коректності програмного коду та файлової структури проекту) та внесення змін у код. Оскільки операція вимірювання метрик є пасивною, то останній метод не використовується.
- Зареєструвати нові рядки, що міститимуть назву операції та її класу, додати значення цих рядків до конфігураційного файлу.

– Зареєструвати саму операцію у файлі plugin.xml пакета org.eclipse.photran.ui.vpg.

Далі наведено приклад текста файла маніфеста для реєстрації рефакторингу:

```
<group>
<submenu name="Code Analysis"><!--
Refactorings to perform code analysis -->
<resourceRefactoring
class="org.eclipse.photran.internal.
corerefactoring.MeasureCodeMetrics-
Refactoring"/>
</submenu>
</group>
```

3. Реалізація модуля вимірювання метрик для Photran

Для отримання доступу до структури програми та вимірювання метрик використовується об'єкт AST. Програмні конструкції, що необхідно враховувати при цьому, у синтаксичному дереві представлені вузлами які, у свою чергу, представлені в програмному інтерфейсі Photran об'єктами класів, що реалізують базовий інтерфейс IASTNode. Обхід дерева здійснюється за допомогою патерна «відвідувач», для чого необхідно реалізувати інтерфейс IASTVisitor. Цей інтерфейс містить методи, в тілі яких має здійснюватися обробка «відвіданого» структурного вузла. Кожен вузол у свою чергу містить метод accept(IASTVisitor), який є реалізацією патерну та надає переданому об'єкту доступ до внутрішніх конструкцій та дочірніх вузлів. Наприклад, клас ASTIfConstructNode відповідає за конструкцію умовного переходу, та в реалізації методу accept дозволяє «відвідувачу» обійти тіло конструкції (код, куди здійснюється умовний перехід), конструкцію else if та конструкцію else.

Тривіальним прикладом структур дерева та відвідувача буде програма, що містить визначення певних даних (змінної, масиву тощо) та виклик підпрограми. Підпрограма, в свою чергу, міститиме конструкцію умовного переходу (if-elseif). Клас-відвідувач для обходу такого дерева має реалізувати ряд методів, по одному для кожного типу структури. Спрощену діаграму класів для такого випадку показано на рис. 1.

Алгоритми вимірювання метрик:

SLOC та BLOC – найпростіші можливі метрики, для вимірювання яких достатньо отримати текст програми за допомогою методу IASTNode.toString(), розбити його на рядки та порахувати їх загальну кількість та кількість порожніх рядків.

Для вирахування метрик LOC, CLOC, що залежать від вмісту рядка, було використано метод IFortranAST. findFirstTokenOnLine(). У випадку наявності «токена» (програмного виразу) рядок є рядком коду, інакше – рядком з коментарем.

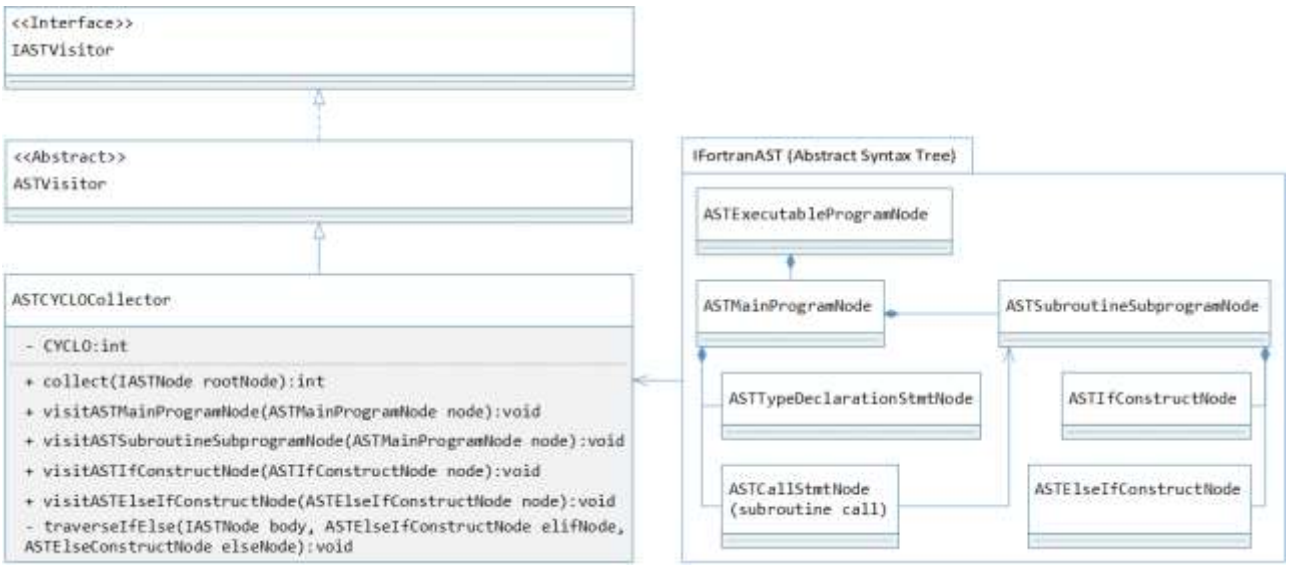

Рис. 1. Спрощена діаграма класів для патерна «відвідувач» у Photran

Для вимірювання **зусиль за Холстедом** необхідно отримати чотири базові значення – загальну та унікальну кількості операндів та операторів (N_1, N_2, n_1, n_2). Кожному операнду та оператору в синтаксичному дереві відповідає певний об'єкт класу Token. У кожного токена, у свою чергу, є тип (представлений об'єктом класу Terminal), що унікально ідентифікує синтаксичні одиниці мови.

Операнди у Fortran діляться на два типи: «ідентифікатори» (змінні та імена функцій або підпрограм), що отримують тип токена T_IDENT та літерали або константи, що отримують тип у залежності від типу константи (T_ICON для цілочисельного літерала, T_SCON для строкового літерала тощо). Літерали, навіть однакові за значенням, завжди вважаються унікальними, тому їх кількість просто сумується і додається до N_2 та n_2. До N_2 також додається сумарна кількість ідентифікаторів, але оскільки одного визначення (однієї змінної) можуть стосуватися багато токенів, то для підрахунку кількості унікальних ідентифікаторів необхідно відслідкувати ці визначення. Для цього VPG надає метод Token.resolveBinding() що вертає унікальний об'єкт визначення (класу Definition), що стосується даного ідентифікатора. Загальна кількість цих об'єктів після обходу всіх токенів додається до n_2.

Всі токени, що не є ідентифікаторами або константами, вважаються операторами. Їх загальна кількість записується в N_1, а кількість унікальних – в n_1. Після цього здійснюються обчислення, описані в розділі 1.

Збір інформації для вирахування **цикломатичної складності** вимагає повного обходу синтаксичного дерева та зміни значення метрики в залежності від типу конструкції. Конструкції, що вимагають обходу «відвідувачем» при обчисленні цикломатичної складності, наведені в табл. 1.

Таблиця 1

Дії при виявленні мовних конструкцій

Назва конструкції	Приклад конструкції	Дія при виявленні
if statement	if () ! action	– збільшення значення метрики на 1

if construct	if () then
	! body
	end if
		– збільшення значення метрики на 1
		– обхід тіла конструкції

else if construct	else if () then ! body end if	– збільшення значення метрики на 1 – обхід тіла конструкції

else construct	else
	! body
	end if
		– обхід тіла конструкції

loop construct	do [label] [while ()] ! body [label | end do]	– збільшення значення метрики на 1 – обхід тіла конструкції
assigned/computed goto statement	goto (label[, label]), value	– збільшення значення метрики на кількість аргументів label
case construct	select case () ! body end select	– обхід тіла конструкції
case statement	case ()	– збільшення значення метрики на 1
forall statement	forall () ! action	– збільшення значення метрики на 1
forall construct	forall () then ! body end forall	– збільшення значення метрики на 1 – обхід тіла конструкції

Особлива ситуація виникає з конструкцією where. Ця конструкція призначена для скороченого запису фільтрації масивів за певною умовою. В найпростішому випадку (один «рівень» фільтрації) препроцесор перетворить твердження where на один цикл do з одним набором умовних переходів всередині. При додаванні додаткового рівня фільтрації, препроцесор вимушений виконати «розгортання» конструкції для того щоб другий рівень фільтрації виконувався після звершення першого (на оновлених даних) [5]. Приклад таких конструкцій наведено в табл. 2.

З цієї причини, звичайний обхід конструкцій where не дасть виміряти коректного значення метрики. Цикломатична складність вузла в цьому випадку залежить не тільки від типу, але й від рівня фільтрації на якому перебуває вузол, загальної вкладеності рівнів фільтрації та їх кількості всередині вузла. Оскільки конструкції where можуть містити в собі тільки інші конструкції where, то зручно розглядати where найвищого рівня як окреме дерево.

Позначимо:

n – кількість рівнів фільтрації,

m_i – кількість вузлів на i-ому рівні, $i = \overline{1,n}$,

w_{ij} – j-ий вузол -го рівня, $i = \overline{1,n}$, $j = \overline{1,m_i}$,

s_{ij} – кількість дочірніх вузлів w_{ij}, $i = \overline{1,n}, j = \overline{1,m_i}$,

c_{ijk} – k-ий дочірній вузол w_{ij}, $i = \overline{1,n}, j = \overline{1,m_i}$,

$L(x)$ – кількість викликів вузла x.

Таблиця 2

Перетворення конструкцій where препроцесором

К-сть шарів фільтрації	Вихідний код	Фактичний код	Значення метрики

1	where (condition1)
	  ! body1
	elsewhere (condition2)
	  ! body2
	elsewhere
	  ! body3
	end where
		do
		  if (condition1) then
		  ! body1
		  else if (condition2) then
		  ! body2
		  else
		  ! body3
		  end if
		end do
			4

2	where (condition1)
	  where (condition3)
	  ! body3
	  end where
	  ! body1
	elsewhere (condition2)
	  ! body2
	end where
		do
		  if (condition1) then
		  ! body1
		  else if (condition2) then
		  ! body2
		  end if
		end do
		
		do
		  if (condition1) then
		  if (condition3) then
		  ! body3
		  end if
		  end if
		end do
			7

Проаналізувавши приклади, наведені в табл. 1, видно що вузол w_{ij} буде задіяно один раз для -го рівня і по одному разу для кожного рівня, в якому лежать його нащадки. З цього виходить, що кількість викликів вузла $L(x)$ на одиницю більша за його висоту у дереві і її можна обчислити за рекурсивною формулою:

$$L(w_{ij}) = \max_k \{L(c_{ijk})\}_{k=1}^{s_{ij}} + 1. \qquad (6)$$

За допомогою формули (6) можна порахувати кількість конструкцій if, які буде створено препроцесором для певної конструкції where. Водночас, оскільки для кожного рівня необхідно окремо обійти масив, кількість циклів, що буде створено, дорівнює висоті дерева. Тому загальну цикломатичну складність такої конструкції можна обчислити як:

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} L(w_{ij}) + \max_j \{L(w_{1j})\}_{j=1}^{m_1}. \qquad (7)$$

Для демонстрації вимірювання метрик використано проект-приклад, завантажений з інтернету. Приклад виводу програми показано на рис. 2.

filename	SLOC	LOC	CLOC	BLOC	CYCLO	EFFORT
...	...	...	...	...	...	...
tavg.f90	1599	806	478	315	115	5046464,17
test_prognostic.f	121	53	39	29	1	25533,89
time_manageme	4319	2018	1370	931	329	12638544,90
timers.f90	876	367	306	203	47	1815981,12
topostress.f90	399	172	138	89	28	913376,87
vertical_mix.f90	1545	749	456	340	107	9218730,17
vmix_const.f90	242	97	85	60	12	292245,27
vmix_kpp.f90	2072	1014	645	413	168	15959973,99
vmix_rich.f90	403	168	136	99	25	1057082,47
xdisplay.f90	215	73	92	50	11	105203,90
Total	64670	32622	18789	13259	5412	314670533,81
Execution took PT5.9464045						

Рис. 2. Приклад виводу програми

При багаторазовому запуску програми середня швидкість оброблення вихідного коду складає ~1000 строк/сек. Вимірювання проводилися з використанням комп'ютера оснащеного процесором Intel Core i7-9750H 2.6GHz з обсягом оперативної пам'яті 16GB.

4. Порівняння розробленого засобу з аналогами

На початку тисячоліття хороший інструмент для рефакторингу має містити швидкий статичний аналізатор синтаксису, потужний модуль пошуку та заміни. На сьогоднішній день цей список буде надто коротким щоб задовольнити всі потреби розробника. При порівнянні сучасних інструментів рефакторингу мова йдеться про їх інфраструктуру, про можливість інтеграції з сучасними інтегрованими середовищами розробки, про перевірений та зручний інтерфейс користувача, гарну масштабованість для можливості підтримки різних розмірів проекту, компіляторів та середовищ виконання, часті оновлення, хорошу документованість, інструменти для роботи з командою, тобто мова йдеться вже про екосистему мови програмування.

За час існування Fortran створена велика кількість засобів для автоматизації навігації та аналізу програмного коду (CASE-інструментів). Вони здебільшого діляться на чотири категорії:

технічні аналізатори вихідного коду. Це програмні засоби, призначені для перегляду тексту програми та пошуку застарілого або «метрвого» коду та синтаксичних помилок [6];

стилістичні аналізатори вихідного коду. Вказують на заплутаність тексту програми, недостатню кількість коментарів та шкідливі стилістичні або архітектурні практики [7];

засоби навігації вихідним кодом. Дозволяють здійснювати переміщення графом потоку даних (між змінними та їх визначеннями) та потоку керування (деревом викликів підпрограм), а також здійснювати пошук за ключовими словами, типами даних тощо [8];

засоби рефакторингу вихідного коду. Надають інструментарій для автоматичної реорганізації тексту програми з метою покращення структури, швидкодії, полегшення внесення змін та подальшої підтримки продукту.

Засоби для вимірювання програмних метрик належать до першої категорії, але оскільки сучасна парадигма інтегрованого середовища програмування передбачає об'єднання більшості необхідних розробникам інструментів в один продукт, ці засоби зазвичай містять функціонал з інших категорій. Найбільш розвинуті CASE-інструменти для Fortran (крім Photran) наведені у табл. 3.

В порівнянні з перерахованими засобами, Photran має ряд переваг. Eclipse IDE в його основі робить його однаково простим та гнучким у налаштуванні на Windows, Linux та MacOS. Відкритий вихідний код та ліцензування за Eclipse Public License означає що будь-хто може переглядати, змінювати та використовувати готові компоненти для реалізації нового функціоналу. Можливість прямої взаємодії з AST означає порівняну легкість реалізації на його основі систем переписувальних правил [9] або графічних візуалізацій [10]. В сукупності з навігацією вихідним кодом Fortran, здійсненням рефакторингів та вимірюванням метрик, Photran є потужним та доступним засобом автоматизації підвищення якості програмного забезпечення в галузях, які найбільше цього потребують.

Таблиця 3

Порівняння CASE-інструментів для Fortran

Назва	Стан підтримки	Загальний опис	Збір програмних метрик
Understand	В активній розробці	Інтегроване середовище, що містить функціонал для навігації вихідним кодом та сфокусоване на створенні візуальних звітів по структурі проекту.	Збирає метрики LOC та інші кількісні метрики (кількість файлів, підпрограм, викликів функцій тощо).
SCSE	В активній розробці	Потужний інструмент для навігації вихідним кодом, основним функціоналом є виконання пошукових запитів на великих об'ємах даних.	Збирає метрики LOC, цикломатичну складність та зусилля за Холстедом.
FPT	Остання активність 07.2020	Технічний аналізатор вихідного коду з фокусом на міграціях програм між різними середовищами виконання та операційними системами.	Збирає метрики LOC та цикломатичну складність.
Forcheck	В активній розробці	Технічний аналізатор вихідного коду, що підтримується вже більше 30 років [11], з фокусом на пошуку неявних помилок у тексті програм.	Збирає метрики LOC та цикломатичну складність.
Fortran Analyzer	Остання активність 09.2019	Стилістичний аналізатор вихідного коду з фокусом на альтернативних метриках, що вказують на «чистоту» коду, таких як глибина вкладених циклів, документованість програмних елементів (файлів, підпрограм, модулів).	Збирає метрику SLOC та інші кількісні метрики (кількість підпрограм, викликів функцій,).

Висновки

В роботі розроблені алгоритми для обходу синтаксичного дерева, на основі яких реалізовано додаток до плагіну Photran для оцінювання якості вихідного коду мови Fortran за допомогою вимірювання програмних метрик. Використання розробленого засобу є більш дієвим у поєднанні з іншими функціями Photran для визначення, усунення та відслідковування динаміки росту програми. Проведено порівняння поточного стану та перспектив розвитку інфраструктури середовища Photran з іншими CASE-інструментами. До подальших напрямків розробки належать створення інструментів для візуалізації структури вихідного коду, рефакторингів для оптимізації паралельних обчислень, засобів для стилістичного аналізу програм.

Література

1. Overbey J.L., Negara S., Johnson R.E. Refactoring and the evolution of Fortran. Workshop on Software Engineering for Computational Science and Engineering, Vancouver, BC. 2009. P. 28–34.
2. Hariprasad T., Vidhyagaran G., Seenu K., Thirumalai C. Software complexity analysis using halstead metrics. International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli. 2017. P. 1109 –1113.
3. McCabe T.J. A Complexity Measure. IEEE Transactions on Software Engineering. 1976. N 4. P. 308–320.
4. Photran Developer's Guide [Електронний ресурс]. Режим доступу до ресурсу: https://git.eclipse.org/c/ptp/org.eclipse.photran.git/plain/org.eclipse.photran-dev-docs/dev-guide/dev-guide-specialized.pdf.
5. Metcalf M., Reid J.K. Fortran 90/95 explained. USA: Oxford University Press, Inc. 1999. 341 p. (2).
6. fpt – Tools for Fortran Engineering [Електронний ресурс]. Режим доступу до ресурсу: http://www.simconglobal.com/fpt_summary.html.
7. García-Rodríguez M., Añel J., Foujols M., Rodeiro J. FortranAnalyser: A software tool to assess Fortran code quality [Електронний ресурс]. 2016. Режим доступу до ресурсу: http://fortrananalyser.ephyslab.uvigo.es/docs/access.pdf.
8. Source Code Search Engine [Електронний ресурс]. Режим доступу до ресурсу: http://www.semanticdesigns.com/Products/SearchEngine/?site=SoftwareRecommendations.
9. Дорошенко А.Ю., Жереб К.А., Туліка Є.М. Розпаралелювання програм на фортрані з використанням техніки переписувальних правил. *Проблеми програмування*. 2012. № 2-3. С. 388–397.
10. Fortran 77 Flowcharting Utility [Електронний ресурс]. Режим доступу до ресурсу: http://www.deater.net/weave/vmwprod/f77_diagram/.
11. Lawden M.D. FORCHECK. A Fortran Verifier and Programming Aid. Lawden. Starlink User Note. 1990. N 73. P. 1–3. Top 10 Digital Transformation Trends For 2019 [Електронний ресурс]. Режим доступу до ресурсу: https://www.forbes.com/sites/danielnewman/2018/09/11/top-10-digital-transformation-trends-for-2019/#17d55d1e3c30.

References

1. Overbey J.L., Negara S., Johnson R.E. Refactoring and the evolution of Fortran. Workshop on Software Engineering for Computational Science and Engineering, Vancouver, BC. 2009. P. 28–34.
2. Hariprasad T., Vidhyagaran G., Seenu K., Thirumalai C. Software complexity analysis using halstead metrics. International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli. 2017. P. 1109–1113.
3. McCabe T.J. A Complexity Measure. IEEE Transactions on Software Engineering. 1976. N 4. P. 308–320.
4. Photran Developer's Guide [online] Available at: <https://git.eclipse.org/c/ptp/org.eclipse.photran.git/plain/org.eclipse.photran-dev-docs/dev-guide/dev-guide-specialized.pdf> [Accessed 12 February 2021].
5. Metcalf M., Reid J.K. Fortran 90/95 explained. USA: Oxford University Press, Inc. 1999. 341 p. (2).
6. fpt – Tools for Fortran Engineering [online] Available at: <http://www.simconglobal.com/fpt_summary.html> [Accessed 12 February 2021].
7. García-Rodríguez M., Añel J., Foujols M., Rodeiro J. FortranAnalyser: A software tool to assess Fortran code quality [online]

Fortrananalyser.ephyslab.uvigo.es. Available at: <http://fortrananalyser.ephyslab.uvigo.es/docs/access.pdf> [Accessed 12 February 2021].
8. Semanticdesigns.com. n.d. Source Code Search Engine. [online] Available at: <http://www.semanticdesigns.com/Products/SearchEngine/?site=SoftwareRecommendations> [Accessed 12 February 2021].
9. Doroshenko A., Zhereb K. and Tulika Y. Parallelization of Fortran programs using rewriting rules. *Problems of programming*. 2012. Vol. 2-3. P. 388–397.
10. Deater.net. n.d. 10. Fortran 77 Flowcharting Utility. [online] Available at: <http://www.deater.net/weave/vmwprod/f77_diagram/> [Accessed 12 February 2021].
11. Lawden M., 1990. FORCHECK – A Fortran Verifier and Programming Aid. Starlink User Note, 73(1). P. 1–3.

Одержано 13.02.2021

***Про автора*:**

Покровський Андрій Максимович,
студент 2 курсу магістратури в
НТУУ “КПІ імені Ігоря Сікорського”

Місце роботи автора:

Національний технічний університет України "КПІ імені Ігоря Сікорського", кафедра автоматики і управління в технічних системах.
проспект Перемоги 37.

E-mail: a.m.pokrovskyi@gmail.com