ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

УДК 004.9 https://doi.org/10.35546/kntu2078-4481.2021.3.9

В.І. ШИНКАРЕНКО
Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна
ORCID: 0000-0001-8738-7225
О.О. ЖЕВАГО
Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна
ORCID: 0000-0003-0019-8320

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОЦЕСІВ НАЛАГОДЖЕННЯ КОМП'ЮТЕРНИХ ПРОГРАМ СТУДЕНТАМИ З ВИКОРИСТАННЯМ PROCESS MINING

Розуміння того, як студенти налагоджують програми та з якими проблемами вони стикаються, має велике значення для підвищення якості навчання програмуванню. Аналізуються процеси налагодження комп'ютерних програм з метою підвищення якості навчання студентів програмуванню. Наведено приклад оцінки поведінки й навичок налагодження студентів. Процес налагодження тексту програми розглядається як послідовність дій при роботі з інструментами в середовищі розробки. Для виявлення навичок налагодження застосовано метод підсіву помилок. Розроблено програми з типовими логічними помилками. Виконано експеримент в якому проведено оцінку дій 41 розробника, що виконує налагодження. В рамках експерименту студенти повинні були виправити типові логічні помилки, використовуючи інтегроване середовище розробки Visual Studio. Відстежувалися дії кожного розробника при виконанні поставлених завдань. Формування моделей процесів налагодження виконано методами Process Mining. На основі отриманих моделей процесів налагодження та найбільш часто повторювальних сесій вдалося виявити 4 шаблони поведінки учасників експерименту. Результати демонструють ефективність Process Mining для кращого розуміння того, як розробники виконують налагодження програм. Виявлено, що функції налагоджувача Visual Studio використовуються незадовільно. Учасники, які не використовували засоби налагодження натомість використовували метод спроб і помилок, а також використовували виведення значень на екран. Ці результати закликають посилити практичний досвід налагодження в навчальних програмах. Потребує подальший розвиток інструментальних та організаційних засобів навчання студентів налагодженню в курсах з програмної інженерії.

Ключові слова: Process Mining, налагодження, Visual Studio, експеримент, шаблон налагодження, навчання програмуванню.

В. И. ШИНКАРЕНКО
Днепровский национальный университет железнодорожного транспортаимени академика В. Лазаряна
ORCID: 0000-0001-8738-7225
А. А. ЖЕВАГО
Днепровский национальный университет железнодорожного транспорта имени академика В. Лазаряна
ORCID: 0000-0003-0019-8320

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ ПРОЦЕССОВ ОТЛАДКИ КОМПЬЮТЕРНЫХ ПРОГРАММ СТУДЕНТАМИ С ИСПОЛЬЗОВАНИЕМ PROCESS MINING

Понимание того, как студенты отлаживают программы и с какими проблемами они сталкиваются, имеет большое значение для повышения качества обучения программированию. Анализируются процессы отладки компьютерных программ с целью повышения качества обучения студентов программированию. Описывается пример оценки поведения и навыков отладки у студентов. Процесс отладки текста программы рассматривается как последовательность действий при работе с инструментами в среде разработки. Для выявления навыков отладки применен метод подсева ошибок. Разработаны программы с типичными логическими ошибками. Проведен эксперимент, в котором оценили действия 41 разработчика, выполняющего отладку. В рамках эксперимента студенты должны были решить типичные логические ошибки, используя интегрированное средство разработки Visual Studio. Отслеживались действия каждого разработчика при выполнении поставленных задач. Формирование моделей процессов отладки выполнено методами Process Mining. На основе полученных моделей процессов отладки и наиболее часто повторяющихся сессий удалось обнаружить 4 шаблоны поведения участников эксперимента. Результаты демонстрируют эффективность Process Mining для

лучшего понимания того, как разработчики подходят к решению задачи отладки. Выявлено, что функции отладчика Visual Studio используются, неудовлетворительно. Участники, которые не использовали средства отладки, вместо этого использовали метод проб и ошибок, а также использовали вывод значений на экран. Эти результаты призывают усилить практический опыт отладки в учебных программах. Требуют дальнейшего развития инструментальные и организационные средства обучения студентов отладке в курсах по программной инженерии.

Ключевые слова: Process Mining, отладка, Visual Studio, эксперимент, шаблон отладки, обучение программированию.

V. I. SHYNKARENKO
Dnipro National University of Railway Transport named after Academician V. Lazaryan
ORCID: 0000-0001-8738-7225
O. O. ZHEVAHO
Dnipro National University of Railway Transport named after Academician V. Lazaryan
ORCID: 0000-0003-0019-8320

EXPERIMENTAL STUDIES OF DEBUGGING PROCESSES OF COMPUTER PROGRAMS BY STUDENTS USING PROCESS MINING

Understanding how students debug programs and what problems they encounter is important for improving the quality of programming instruction. The processes of debugging computer programs are analyzed in order to improve the quality of students' programming instruction. An example of assessing students' debugging behavior and skills is described. The process of debugging program text is considered as a sequence of actions when working with tools in the development environment. To identify the debugging skills, the method of error-sowing is applied. Programs with typical logical errors were developed. An experiment was conducted in which the actions of 41 developers performing debugging were evaluated. As part of the experiment, students had to solve typical logical errors using the Visual Studio integrated development environment. The actions of each developer in completing the tasks were tracked. Debugging process models were generated using Process Mining methods. Based on the obtained models of debugging processes and most frequent sessions it was possible to identify 4 patterns of behavior of the experiment's participants. The results demonstrate the effectiveness of Process Mining for a better understanding of how developers approach the debugging task. The Visual Studio debugger features were found to be used, unsatisfactorily. Participants who did not use the debugging tools instead used trial and error and used the display of values on the screen. These results call for more hands-on debugging experiences in the training programs. The instrumental and organizational tools for teaching students to debug in software engineering courses require further development.

Keywords: Process Mining, debugging, Visual Studio, experiment, debugging pattern, programming learning.

Постановка проблеми

Налагодження – одне з найважливіших, складних і трудомістких завдань при розробці програмного забезпечення. Налагодження та програмування йдуть рука об руку, оскільки малоймовірно, що будь-який програміст напише ідеальну програму з першого разу. Проблеми, з якими новачок стикається вперше, швидше за все, будуть виникати знову і знову, в різних середовищах і на різних мовах програмування. Дуже важливо, щоб студент навчився досліджувати помилки. Розробники знають, що налагодження може забрати довгі години роботи, іноді тільки для зміни одного рядка коду. Це особливо актуально для програмістів початківців, які часто виявляються в розгубленості, коли програма працює не так, як очікувалося. Розробники зазвичай витрачають не менше 30 % свого часу на налагодження і використовують для цього середовище розробки [1].

В основі дослідження лежить ідея про те, що інструменти, які збирають дані про використання середовища розробки, дають більш детальне розуміння роботи розробників. Аналізуючи, як програмісти використовують середовище розробки, можна виявити закономірності поведінки та визначити проблеми, з якими вони стикаються. Процес налагодження тексту програми розглядається як послідовність дій при роботі з інструментами в середовищі розробки. Підхід, спочатку запропонований в [2], фіксує події, пов'язані з налагодженням, яке виконується в середовищі розробки. Розширення середовища розробки для збору даних про використання забезпечує більш тонке розуміння роботи розробників.

Підвищення якості налагодження програмного забезпечення серед студентів представляє великий інтерес в області комп'ютерної освіти. Цього можна досягти шляхом застосування підходів аналітики навчання для вивчення поведінки різних людей при налагодженні програмного забезпечення. Щоб зрозуміти, як насправді виконувався процес, використовуються методи Process Mining. Process Mining – це сімейство методів, спрямованих на аналіз даних про виконання бізнес-процесів, записаних в журналах подій [3]. В якості вхідних даних для методів Process Mining використовуються події, зібрані в

середовищі розробки під час сесій налагодження. Покажемо ефективність Process Mining для кращого розуміння того, як розробники підходять до налагодження програмного забезпечення.

Для підвищення якості навчання студентів програмуванню досліджено і вирішено актуальну науково-практичну задачу аналізу процесів налагодження комп'ютерних програм. Розуміння того, як студенти розробляють та налагоджують програмне забезпечення, і проблем, з якими вони стикаються, має велике значення для підвищення якості навчання програмування.

Аналіз останніх досліджень і публікацій

Методи Process Mining вже використовувалися для вивчення процесів розробки програмного забезпечення, починаючи з журналів, отриманих з систем контролю версій [4] і засобів розробки [5]. Але залишилися невирішеними питання, пов'язані з процесом налагодження комп'ютерних програм. Метою Process Mining є виявлення, моніторинг і поліпшення процесів за допомогою даних з журналу подій інформаційної системи. Process Mining застосовує спеціалізовані алгоритми інтелектуального аналізу даних до журналу подій [3]. Поточне дослідження застосовує методи Process Mining в області програмної інженерії для виявлення навичок налагодження програмного забезпечення. Ґрунтуючись на методах Process Mining, аналізуючи дані про використання середовища розробки, отримуємо шаблони поведінки програмістів.

Проводилося дослідження в якому для візуалізації процесу розробки програмного забезпечення використовувалися засоби розширення середовища розробки Visual Studio [6]. Розроблене розширення складається з трьох модулів: зберігання історії написання програми; обчислення різниці між версіями програми та візуалізації історії розробки. В [7] запропоновано використовувати сучасний огляд коду для оцінки робіт студентів. Сучасний огляд коду повинен включати не тільки перегляд коду, але і огляд процесу розробки програмного забезпечення. Для цього необхідно фіксувати взаємодію програміста із засобом розробки для отримання даних про процеси розробки та налагодження. Інструменти для збору даних про використання середовища розробки забезпечують більш детальне розуміння роботи програмістів, ніж це було можливо раніше [8]. Наразі представляється новий погляд на аналіз процесу налагодження з використанням Process Mining, який дозволяє виявити фактичні процеси, яким слідують програмісти на основі отриманих даних взаємодії з середовищем розробки.

Ряд дослідників в галузі освіти вивчали звички студентів в розробці програм в процесі роботи над завданнями з програмування. Вивчали поведінку студентів, реєструючи компіляції та пов'язані з ними дані [9]. В результаті змогли визначити, які типи помилок студенти найчастіше допускають і як часто вони їх роблять. Збір даних здійснювався за допомогою розширення інтерактивного середовища розробки, яке було встановлено на машинах студентів. Вивчені вісім найбільш поширених помилок початківців [10]. Написано вісім коротких програм, кожна з яких представляла один тип помилки, а учасники експерименту повинні були зіставити програму з типом виявленої помилки. В експерименті брали участь 59 студентів другого курсу Кентського державного університету (США). Експеримент був розроблений таким чином, щоб відстежувати порядок, в якому учасник виправляв помилки, а також час, витрачений на кожну з помилок. Отримані результати дозволяють виявити складні для налагодження помилки. Однак вони не дають можливості оцінити рівень навичок студентів і не визначають, які методи та інструменти налагодження були використані. На основі цих досліджень про найбільш поширені помилки початківців були розроблені програми для проведення експерименту з виявлення навичок налагодження.

Вивчали практику розробників налагодження професійного програмного забезпечення, шляхом відстеження дій восьми розробників в чотирьох різних компаніях протягом дня, спостерігаючи за їх методами, даючи їм подумати вголос і відповісти на кілька запитань [11]. Результати показали, що жоден з розробників не мав спеціальної освіти або підготовки в області налагодження. Також виявили, що жоден з розробників не знав про більш функціональні можливості, такі як умовні точки зупину. Разом з тим майже немає кореляції між використовуваною мовою програмування та інструментами налагодження. Попри те, що наші дослідження значно відрізняються за чисельністю, та методологією, вдалося повторити більшість ключових результатів: широке використання операторів виведення на екран, дуже рідке використання просунутих інструментів і функцій налагодження.

Дослідники розробили систематичні описи процесу налагодження і рекомендації щодо скорочення часу, який програмістам доводиться витрачати на пошук і усунення дефекту, що викликає збій програми [12, 13]. Під час аналізу експерименту досліджується чи використовують студенти стратегії налагодження.

Систематизація результатів досліджень свідчить про нестачу знань про те, як програмісти насправді налагоджують та усувають проблеми в процесі розробки та налагодження програмного забезпечення. Тому для ідентифікації та розуміння поведінки розробників під час налагодження програм пропонується застосовувати методи Process Mining.

Все це дає підстави стверджувати, що доцільним є проведення дослідження, присвяченого вивченню процесів налагодження студентів для виявлення шаблонів поведінки. Розуміння того, як

студенти налагоджують програми та з якими проблемами вони стикаються, має велике значення для підвищення якості навчання програмуванню.

Мета дослідження

Мета роботи експериментально дослідити процес налагодження студентами поширених логічних помилок. Щоб виявити шаблони процесів налагодження, які сприяють розумінню поведінки програміста під час налагодження в середовищі розробки. За допомогою цього вдасться вдосконалити процес навчання студентів основам програмування, шляхом підвищення контролю за процесом виконання робіт та внаслідок підтримки індивідуального підходу до навчання.

Для досягнення мети були поставлені такі завдання:

– розробити програми із заздалегідь визначеними помилками на основі досліджень про найбільш поширені помилки початківців [9, 10, 14];

– провести експеримент з налагодження, під час якого за допомогою розроблених інструментів [2] відстежувати та фіксувати дії учасників в середовищі розробки Visual Studio та формувати журнали подій;

– на основі журналів подій сформувати за допомогою методів Process Mining моделі процесів налагодження;

– виявити шаблони дій під час налагодження.

Викладення основного матеріалу дослідження

Засоби та методи дослідження

Виконано експериментальне дослідження процесів налагодження програм студентами різних курсів спеціальностей 121 «Програмна інженерія» та 123 «Комп'ютерна інженерія».

Експеримент проведено у вигляді олімпіади з налагодження у Дніпровському національному університеті залізничного транспорту імені академіка В. Лазаряна (Дніпро, Україна).

Для виявлення навичок налагодження застосовано метод підсіву помилок.

Експериментально досліджувався процес налагодження студентами поширених логічних помилок. Логічна помилка виникає, коли код синтаксично правильний, але програма виконується неправильно. Цей тип помилок, як правило, досить складний і студентам важче знайти та виправити його.

Для проведення експерименту розроблено тридцять коротких програм на мові програмування C++, кожна з яких включає один з 15 найбільш розповсюджених типів на основі попередніх досліджень [9, 10, 14]. Уже з першого курсу студенти мають навички програмування та налагодження на мові C++, тому саме вона була обрана. Для кожної програми логічна помилка проявляється одним рядком коду. Тобто помилка може бути виправлена шляхом зміни тільки одного рядка коду. Всі програми синтаксично коректні, але виконання кожної з них призводить до некоректного результату. Щоб допомогти виявити всі помилки, студентам надається зразок вхідних та вихідних даних.

Процес налагодження тексту програми розглядається як послідовність дій при роботі з інструментами в середовищі розробки. Для збору та аналізу поведінки під час налагодження розроблено розширення до Visual Studio, яке відстежує всі дії при роботі з середовищем розробки [2] та фіксує їх в журналах подій.

Формування моделі процесу дозволяє отримати спосіб і порядок виконання процесу. Для виявлення моделі процесу налагодження з журналу подій використовується ProM. ProM – це широко поширений фреймворк з відкритим кодом, який є де-факто стандартом, для реалізації Process Mining [3]. Використовується підхід заснований на побудові Directly Follow Graph (DFG) з журналів подій. DFG представляє кожну подію у вигляді прямокутника і з'єднує дві події, якщо одна з них безпосередньо слідує за іншою. Більш того, кожне ребро має вагу, яка вказує на кількість його входжень в журнал. Даний підхід гарантує, що виявлена модель є коректною.

Систематичне знайомство студентів з різними типами помилок може значно поліпшити навички налагодження. Крім того, ізольоване виявлення окремих помилок спрощує пошук і усунення проблеми. Студентам пропонується визначити помилку та усунути її в установлений термін. Аналізуючи стратегії усунення помилок, виявлялися шаблони, які впливають або пояснюють поведінку при налагодженні і які необхідно враховувати при навчанні.

Розроблені програми із заздалегідь визначеними помилками

Розроблено програми з типовими логічними помилками. Для експерименту обрано п'ятнадцять різних типів помилок, наведених у табл. 1.

Таблиця 1

Типи логічних помилок використаних в експерименті

Номер	Тип	Опис
1	Спроба встановити значення змінній більше за максимально допустиме значення для відповідного типу	Виникає коли програма намагається записати до змінної значення, яке більше за максимальне значення визначеного типу
2	Ділення цілого числа на ціле число, тому результат округляється	Виникає коли обидва операнда є цілими числами та результат ділення дає дробовий результат, але без оператора приведення до типу з плаваючою комою, тому дробова частина буде відкинута
3	Інкремент не значення, а вказівника	Виникає коли програма намагається збільшити змінну, на яку посилається вказівник (*counter++;), так як пріоритет операції «++» вище, ніж в операції «*» (розіменування вказівника)
4	Неправильне використання функції sizeof	Виникає коли намагаються використовувати sizeof для масиву переданого у якості параметра функції, адже в такому випадку отримуємо розмір вказівника, а не кількість елементів масиву
5	Помилка в написанні	Виникає коли помиляються в написанні операторів, наприклад «=+» замість «+=»
6	Помилка при перестановці значень	Виникає коли пропускають використання тимчасової змінної для заміни значень двох змінних, або роблять це неправильно
7	Помилка паралелізму	Виникає коли міняють місцями твердження програми
8	Неправильне використання оператора switch, відсутність оператора break	Виникає коли забувають оператор break в операторі switch, що призводить до некоректного виконання програми
9	Зайва крапка з комою	Виникає коли додають зайву крапку з комою, наприклад, крапка з комою після оператору циклу, в такому випадку цикл не має тіла і викликає неправильне функціонування програми
10	Неправильна послідовність арифметичних операцій	Виникає коли програміст не розуміє пріоритети операцій і в результаті отримує помилкові значення обчислень
11	Відсутність фігурних дужок	Виникає коли забувають вказати фігурні дужки
12	Нерозуміння області дії змінних	Виникає коли створюють декілька змінних з однією назвою в рамках однієї функції
13	Відсутність ініціалізації	Виникає коли забувають ініціалізувати змінну перед тим як її використовувати, що призводить до виключних ситуацій під час виконання
14	Використання одинарного знаку рівності	Виникає при використанні одинарного знаку рівності «=» для перевірки рівності в умовах замість подвійного знаку рівності «==»
15	Помилка з граничними умовами	Виникає коли програміст виконує ітерацію циклу більшу або меншу кількість разів, ніж очікується

Типи обрані тому, що вони часто зустрічаються серед студентів та є особливо важкими для виявлення програмістами початківцями на основі попередніх досліджень [9, 10, 14].

Результати експерименту з налагодження програм

У експерименті взяв участь 41 студент з першого по п’ятий курси. Всі вони мали попередній досвід розробки на C подібних мовах програмування і використання середовища розробки Visual Studio. Всього студентам треба було виконати п’ятнадцять завдань за 4 години.

На основі кількості правильно виконаних завдань учасники були класифіковані як "High", "Middle", "Low". Студенти відносяться до класу "Low", якщо вони виконали менше половини завдань, таких – 9. Студенти з "Middle" класу виконали більше половини завдань, але не всі, таких – 23. Студенти з "High" класу виконали всі завдання, таких – 9.

Тридцять два (~78 %) студента змогли вирішити половину або більше завдань. Дев’ять студентів виправили всі помилки. В середньому їм на це знадобилося трохи більше 2 годин. Жодна з категорій

помилок не є повністю нерозв'язаною усіма студентами. Помилки з категорій 2, 8 та 14 вирішили усі студенти. Немає жодної категорії, яка б містила більше неправильних відповідей, ніж правильних.

Щодо того, які помилки є найбільш складними для студентів, результати показують, що більшість учасників здатні правильно відповісти на половину або більше завдань. Виходячи з кількості правильних відповідей, більшість помилок потрапляли в діапазон від 75 % правильних. Результати свідчать про те, що студенти мають помірні труднощі з категоріями помилок 4 (неправильне використання функції sizeof) та 12 (нерозуміння області дії змінних). 25 % завдань з цих категорій були вирішені неправильно.

В табл. 2 наведено перелік найбільш часто використовуваних команд середовища розробки Visual Studio учасниками експерименту.

Таблиця 2

Частота використання команд під час експерименту

Номер	Команда	Частота	Відсоток використання серед учасників		
			High	Middle	Low
1	DebugStart	2415	15.368 %	14.426 %	14.059 %
2	DebugEnd	2415	15.368 %	14.426 %	14.059 %
3	BuildDone	2415	15.368 %	14.426 %	14.059 %
4	BuildBegin	1433	8.893 %	8.195 %	9.855 %
5	CodeChanged	1291	7.86 %	7.074 %	10.096 %
6	BreakpointHitted	323	0.821 %	2.433 %	2.068 %
7	DocumentOpened	248	1.431 %	1.718 %	0.896 %
8	Edit.Undo	230	0.845 %	1.12 %	3.067 %
9	BreakpointAdd	168	0.375 %	1.472 %	0.482 %
10	BreakpointRemove	160	0.634 %	1.344 %	0.241 %
11	Edit.Paste	114	0.446 %	0.896 %	0.379 %
12	StepOver	101	0.211 %	0.704 %	0.896 %
13	ExceptionThrown	90	0.634 %	0.63 %	0.138 %
14	StepInto	85	0 %	0.587 %	1.034 %
15	Edit.Copy	74	0.352 %	0.566 %	0.207 %

Відносно невеликий набір команд зустрічається майже у всіх учасників, а саме дев'ять команд зустрічаються у 90 % або більше студентів. Це загальний набір команд: запуск і зупинка налагоджувача, зміна коду, виконання програми, відкриття файлів з текстом програми, а також копіювання та вставка. Більшість інших команд зустрічаються у меншого кола розробників. До них відносяться команди такі як установка, видалення і зупинка на точках зупинки, покрокове виконання, відміна зроблених змін, а також події виключних ситуацій. Більшість з цих команд мають широке застосування і повинні бути рекомендовані іншим розробникам.

У табл. 3 наведено статистику практик налагодження, які використовувалися студентами під час експерименту.

Таблиця 3

Статистика використання практик налагодження

Практика	Відсоток сесій в яких застосовано практику		
	High	Middle	Low
Зміна коду	43.82 %	37.13 %	57.84 %
Покрокове виконання	1.37 %	8.58 %	12.75 %
Точки зупину	5.34 %	17.31 %	14.71 %
Покрокове виконання серед сесій з точками зупину	25.71 %	49.57 %	86.67 %
Виведення на екран	4.83 %	7.33 %	17.41 %

Як можна побачити учасники з класу "Low" роблять набагато більше змін коду ніж інші при цьому у них значно менше сесій налагодження. Ще одна виразна особливість учасників класу "Low" це частіше використання практики виведення на екран значень змінних аніж встановлення точок зупину чи покрокове виконання.

Також в табл. 3 можна помітити два типи використання точок зупину. По-перше, для перевірки чи виконується частина коду, а по-друге, для покрокового виконання програми. При цьому лише учасники класу "Low" в переважній більшості використовують точки зупину для покрокового виконання (86.67 %). Учасники з класу "High" навпаки лише у 25.71 % сесій використовували таким чином. Учасники з класу "High" взагалі майже не використовували практики встановлення точок зупину та

покрокового виконання. Це може вказувати на те, що їх навичок розробника достатньо щоб одразу робити правильні гіпотези, а іншим необхідно використовувати налагодження для вивчення поведінки програми.

Моделі процесів налагодження засобами Process Mining

Щоб краще зрозуміти, як насправді виконувався процес налагодження використовуються методи Process Mining. Вхідними даними для алгоритмів Process Mining є журнали подій, сформовані під час сесій налагодження в інтегрованому засобі розробки Visual Studio. Приклад такого журналу наведено на рис. 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext" />
  <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext" />
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext" />
  <classifier name="Activity" keys="concept:name" />
  <string key="concept:name" value="I Doe Task002.xes" />
  <trace>
    <string key="developer" value="I Doe" />
    <string key="project" value="Task002" />
    <date key="finishedDateTime" value="2021-05-21T09:02:52.5218810+03:00" />
    <date key="startedDateTime" value="2021-05-21T09:02:13.0078042+03:00" />
    <string key="concept:name" value="I Doe Task002 Session1" />
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:13.0078042+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="Start" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:13.0078042+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="DocumentOpened" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:40.4171154+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="BuildBegin" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:44.7372113+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="BuildDone" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:47.8993203+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="DebugStart" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:52.5218810+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="DebugEnd" />
    </event>
    <event>
      <date key="time:timestamp" value="2021-05-21T09:02:52.5218810+03:00" />
      <string key="lifecycle:transition" value="complete" />
      <string key="concept:name" value="Finish" />
    </event>
  </trace>
</log>
```

Рис. 1. Приклад журналу подій в форматі XES

Журнал подій структурований як набір записів, де кожна із записів є ланцюжком дій, створених в результаті одного виконання процесу. Журнал подій формується в форматі IEEE eXtensible Event Stream (XES) [15], щоб застосувати методи Process Mining для автоматичного аналізу. На основі таких журналів подій методи Process Mining витягають корисну інформацію, яка дозволяє отримати уявлення

про процес, сформулювати і перевірити гіпотези. Результати Process Mining можуть бути починаючи від повної моделі процесу, до опису основних (частих) шляхів або його відхилень.

Під час експерименту було сформовано 487 файлів журналів подій для 41 учасника. Загалом журнали подій складаються з 2415 сесій налагодження та 16536 подій.

Щоб зрозуміти, що сталося в кожному класі учасників, сформували відповідні моделі процесів на основі журналів подій (фаза виявлення процесу). Зокрема, Directly Follow Graph плагін фреймворку ProM (метод Process Mining) приймає на вхід журнали учасників з "High", "Middle", "Low" класів та створює відповідні моделі процесів, у вигляді DFG.

На рис. 2 зображено загальну модель процесу налагодження комп'ютерних програм учасниками експерименту.

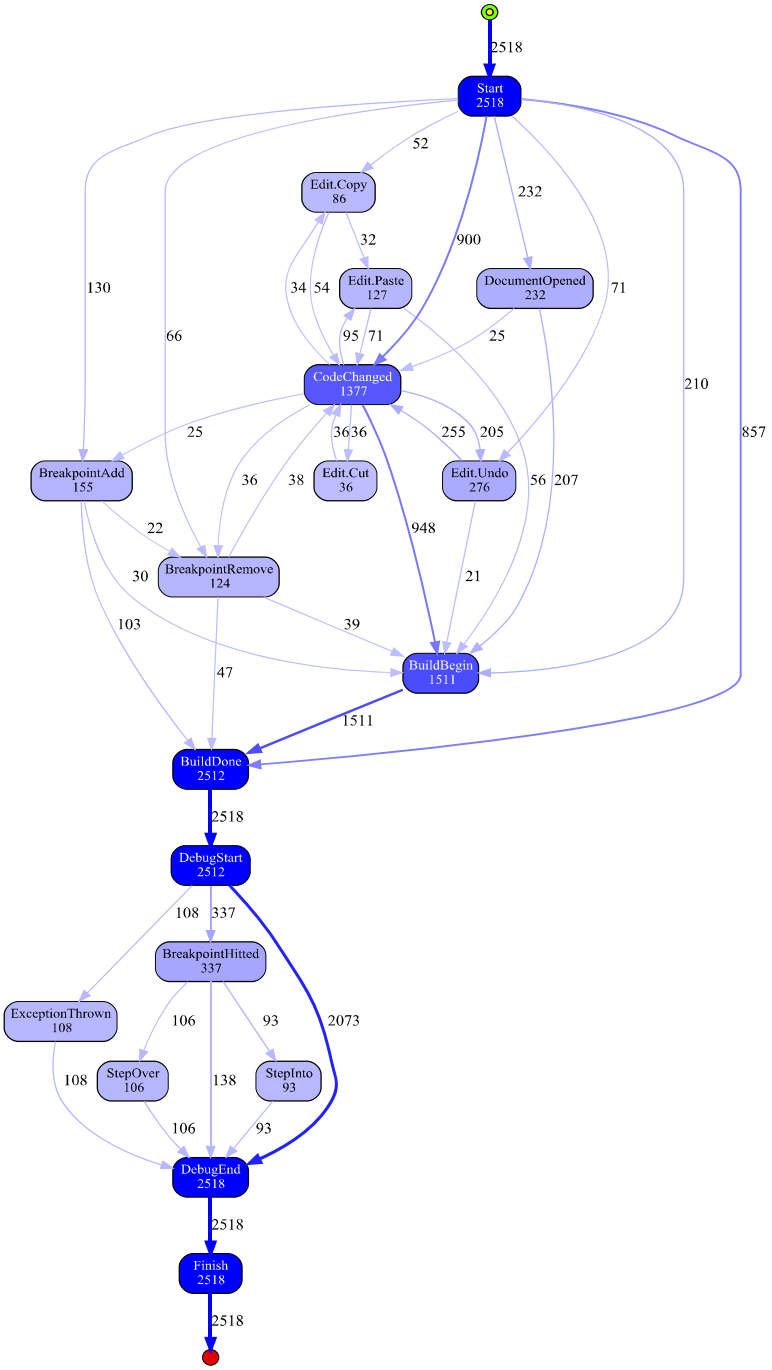


Рис. 2. Загальна модель процесу налагодження у вигляді Directly Follows Graph

Сформовано один загальний файл журналу подій на основі усіх сесій налагодження. Задля того щоб модель була інформативною було проведено фільтрацію засобами ProM. В результаті модель сформовано на основі файлу журналу подій, який складається з 90 % подій, які були відстежені з

середовища розробки Visual Studio під час експерименту. Модель відображає команди, які були використані під час налагодження та зв'язки між ними.

Виявлені шаблони налагодження

На основі отриманих моделей процесів налагодження та найбільш часто повторювальних сесій вдалося виявити 4 шаблони поведінки учасників експерименту.

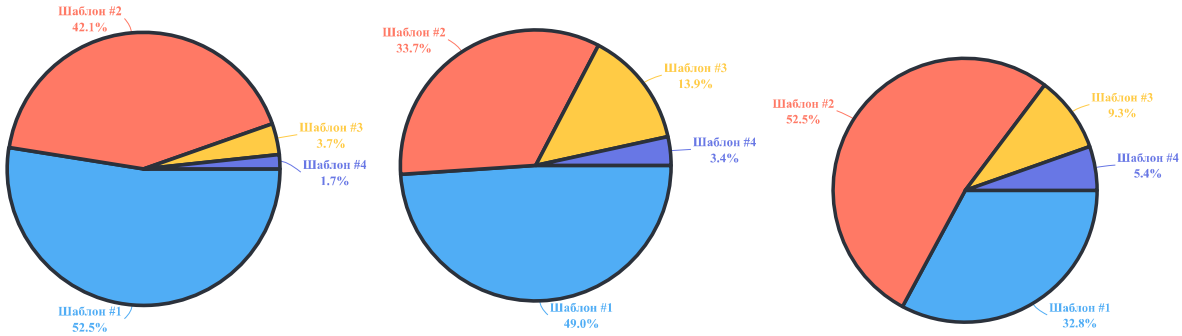
Шаблон #1 – виконання програми в режимі налагодження без будь-яких змін та налагоджувальних дій. Використовується, щоб перевірити поведінку програми.

Шаблон #2 – це зміна тексту програми й виконання в режимі налагодження без будь-яких змін та налагоджувальних дій. Цей шаблон відповідає застосуванню методу спроб та помилок.

Шаблон #3 – це виконання програми в режимі налагодження з встановленими точками зупину. Використовується, щоб дослідити поведінку програми, а саме які її частини виконуються. А також використовується для покрокового виконання.

Шаблон #4 – це зміна тексту програми, встановлення точок зупину й виконання в режимі налагодження. Цей шаблон відповідає застосуванню методу припущення про помилку. Коли робиться припущення про помилку, в цьому місці виконується зміна коду і встановлюються точки зупину для перевірки, що помилка виправлена.

На рис. 3 наведено частоту використання шаблонів різними класами учасників експерименту.



а *б* *в*

Рис. 3. Статистика використання шаблонів учасниками: а – клас "High"; б – клас "Middle"; в – клас "Low"

Як можна побачити на рис. 3 у всіх класах учасників переважає використання перших двох шаблонів, та дуже незначне використання шаблонів #3 та #4. Вагома різниця лише в тому, що в учасників з класу "Low" значно переважає використання методу спроб та помилок, а учасники класу "High" використовують шаблони #3 та #4 лише в 5.4 % сесій налагодження.

На рис. 4–7 наведено, як виявлені шаблони відображаються на отриманих засобами Process Mining моделях процесів налагодження.

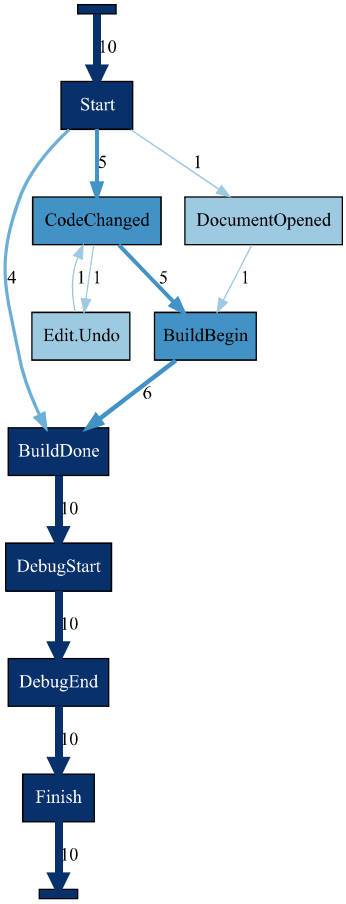


Рис. 4. Модель процесу налагодження із застосуванням методу спроб та помилок

Рис. 4 відображає модель застосування методу спроб та помилок, яке здійснюється через внесення змін (CodeChanged) та виконання програми без будь-яких налагоджувальних дій. Такий процес повторюється допоки не буде знайдено рішення або закінчаться ідеї. Такий метод застосовується усіма класами учасників і різниця лише в тому що учасникам класу "Low" треба більше спроб і вони мають гірший результат.

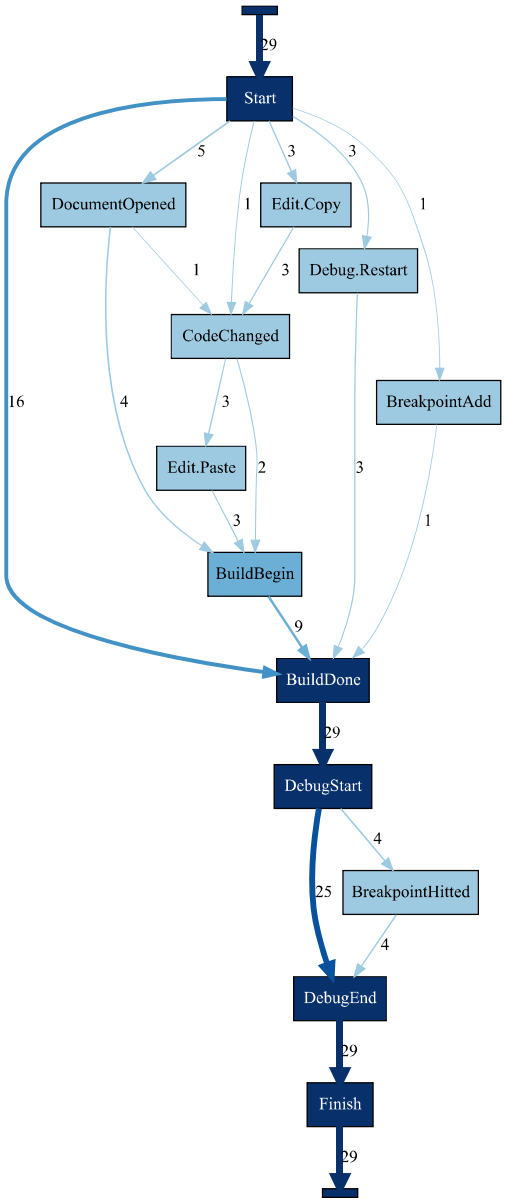


Рис. 5. Модель процесу налагодження із застосуванням точок зупину, але без покрокового виконання

Модель на рис. 5 відображає застосування шаблону #3 при налагодженні, а саме послідовність дій по виконанню програми без будь-яких змін з встановленими точками зупину. Коли точка зупину спрацює є декілька варіантів подальших дій. Перший, закінчити налагодження, в такому випадку мета була переконатися, що частина коду виконується. Другий, продовжити налагодження шляхом покрокового виконання. Як показали результати експерименту другий варіант використовується переважною більшістю учасників з класу "Low". Учасники з класу "Middle" однаково використовують обидва методи, а учасники класу "High" надають перевагу першому, адже вони здатні без покрокового виконання зрозуміти поведінку програми.

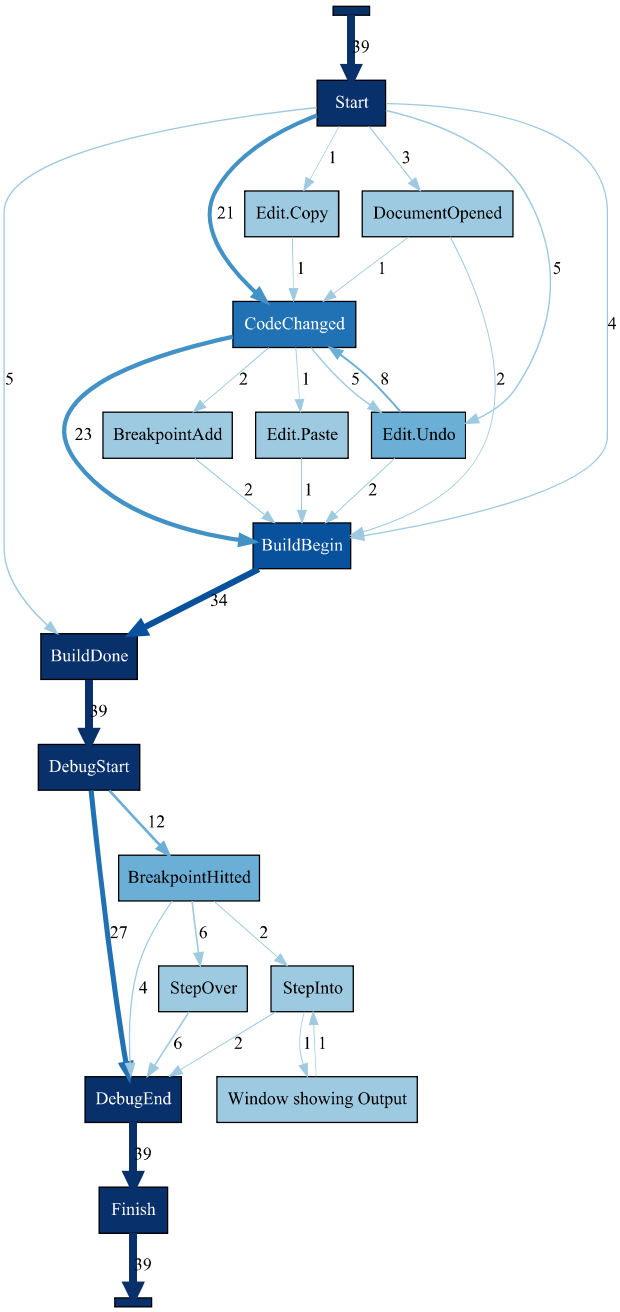


Рис. 6. Модель процесу налагодження застосовуючи шаблон #4

Модель на рис. 6 відображає застосування методу припущення про помилку. Коли програміст висуває гіпотезу, робить зміну тексту програми, щоб виправити помилку, після чого виконує програму з встановленими точками зупину і перевіряє шляхом покрокового виконання чи помилка виправлена.

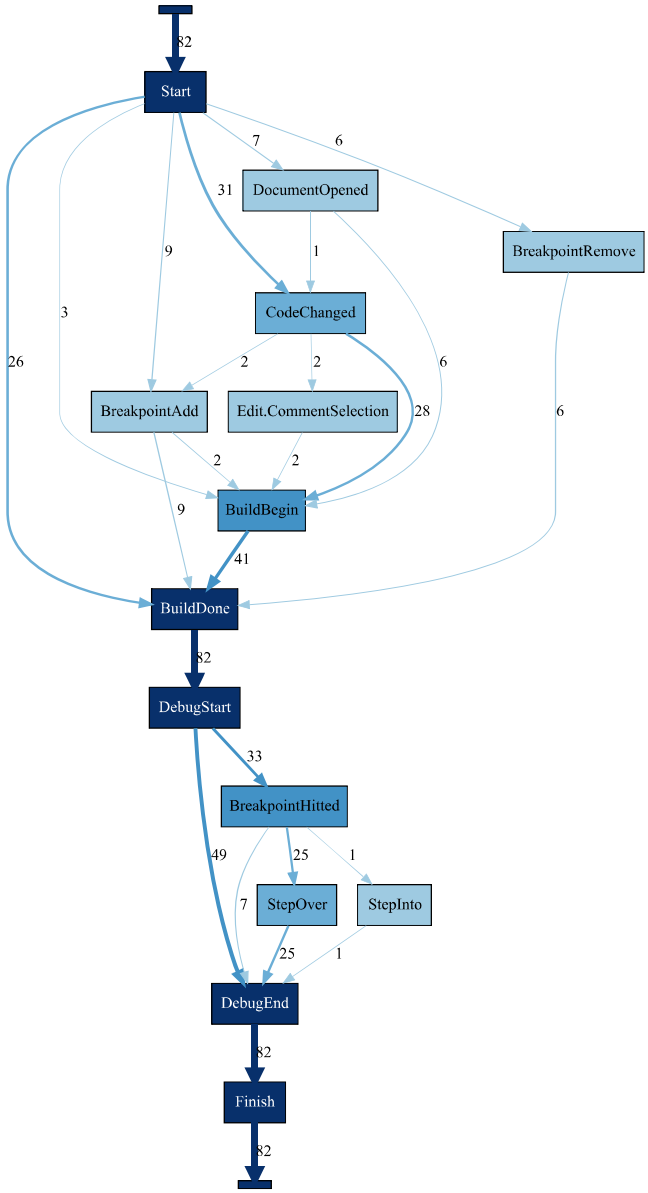


Рис. 7. Модель процесу налагодження із застосуванням одразу усіх шаблонів

На рис. 7 зображена модель одного з учасників експерименту, яка відображає застосування одразу усіх шаблонів під час вирішення завдання. Шаблон #1 – *Start–BuildDone–DebugStart–DebugEnd–Finish*. Шаблон #2 – *Start–CodeChanged–BuildBegin–BuildDone–DebugStart–DebugEnd–Finish*. Шаблон #3 – *Start–BreakpointAdd–BuildDone–DebugStart–BreakpointHitted–DebugEnd–Finish*. Шаблон #4 – *Start–CodeChanged–BreakpointAdd–BuildBegin–BuildDone–DebugStart– BreakpointHitted– StepOver–DebugEnd–Finish*.

Результати демонструють ефективність Process Mining для кращого розуміння того, як розробники налагоджують програми. Виходячи з результатів дослідження, адаптивні методи навчання можна застосовувати для студентів з різним ступенем успішності.

Обговорення результатів експериментального дослідження процесів налагодження комп'ютерних програм студентами

Використання методів Process Mining представляють новий підхід до вивчення процесів налагодження програм. В ході експерименту проведено оцінку розробників, щоб виділити тих, хто домігся найкращих результатів. Потім порівняти поведінку, отриману від цієї групи розробників, з поведінкою інших. Результати показують різні моделі поведінки розробників з різними навичками.

Установка точок зупину і проходження по коду виконувалося більшістю розробників за допомогою налагоджувача, інші функції налагодження використовуються набагато рідше і набагато

меншою кількістю розробників. Лише 2 учасників використовували такі засоби як перегляд значень змінних. Учасники, які не використовували засоби налагодження натомість використовували метод спроб і помилок, а також використовували виведення значень на екран. Ці результати закликають посилити практичний досвід налагодження в навчальних програмах.

Хоча це не дивний результат, але експеримент наочно продемонстрував, що досвід є ключовим фактором навичок налагодження. Адже всі студенти 4 та 5 курсів виконали половину або більше завдань. А серед студентів першого курсу немає таких, які б виконали всі завдання.

Аналіз даних дозволив виявити студентів, які вирішували завдання не самостійно. Одного учасника було дискваліфіковано через те, що всі його завдання були вирішені шляхом вставки готового рішення. Окрім того ще 5 учасникам було не зараховано правильно виконані завдання через відсутність процесу досягнення результату, що викликає велику підозру в не самостійності. Також такі рішення співпадали з рішеннями інших учасників. Моделі процесів таких учасників складалися лише з декількох сесій налагодження, в кожній з яких було не більше 5 дій, одна з яких це команда Edit.Paste великого шматку коду.

Необхідно враховувати при інтерпретації результатів, що дані взяті з обмеженої вибірки, що складається з 41 студента з першого по п’ятий курси Дніпровського національного університету залізничного транспорту імені академіка В. Лазаряна (Дніпро, Україна). Тому учасники мають відносно вузьку і однорідну підготовку в області розробки та налагодження програмного забезпечення.

В подальшому для більш глибокого вивчення було б слушно ускладнити завдання збільшивши кількість помилок та рядків коду, що потребують виправленню. З потреб покращення статистики слід розширяти контингент студентів з різних університетів.

Висновки

1. Розроблено програми з типовими логічними помилками на основі досліджень про найбільш поширені помилки початківців.

2. Проведено експеримент з дослідження процесів налагодження, під час якого за допомогою розроблених інструментів відстежувалися та фіксувалися дії учасників в середовищі розробки. Для виявлення навичок налагодження застосовано метод підсіву помилок. Результати показали навички студентів по налагодженню програм з типовими логічними помилками. Вдалося виявити проблеми, які є більш складними для локалізації та виправлення. Виходячи з результатів дослідження, адаптивні методи навчання можна застосовувати для студентів з різним ступенем успішності. Результати проведених досліджень та впровадження запропонованих процесів та інструментів дозволять підвищити якість навчання студентів внаслідок моніторингу їх роботи.

3. Розроблені моделі процесів налагодження програм засобами Process Mining, що на відміну від існуючих дають можливість автоматизувати аналіз цих процесів.

4. На основі сформованих моделей за допомогою інструментів та методів Process Mining вдалося виявити шаблони дій під час налагодження. Результати дослідження показують, що методи Process Mining корисні для розуміння того, як програмісти виконують налагодження, які методи використовують та з якими труднощами стикаються.

Список використаної літератури

1. LaToza, T. D., & Myers, B. A. (2010). Developers ask reachability questions. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – ICSE'10 (Vol. 1, pp. 185–194). https://doi.org/10.1145/1806799.1806829

2. Shynkarenko, V., & Zhevaho, O. (2020). Development of a toolkit for analyzing software debugging processes using the constructive approach. Eastern-european Journal of Enterprise Technologies, 5(2 (107)), 29–38. https://doi.org/10.15587/1729-4061.2020.215090

3. van der Aalst, W. (2012). Process mining: Overview and opportunities. ACM Transactions on Management Information Systems, 3(2), 1–17. https://doi.org/10.1145/2229156.2229157

4. Rubin, V., Günther, C. W., van der Aalst, W. M. P., Kindler, E., van Dongen, B. F., & Schäfer, W. (2007). Process Mining Framework for Software Processes. In Software Process Dynamics and Agility (pp. 169–181). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-72426-1_15

5. Ardimento, P., Bernardi, M. L., Cimitile, M., & Maggi, F. M. (2019). Evaluating Coding Behavior in Software Development Processes: A Process Mining Approach. In 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). IEEE. http://doi.org/10.1109/icssp.2019.00020

6. Shynkarenko, V., & Zhevago, O. (2019). Visualization of program development process. In 2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT). IEEE. http://doi.org/10.1109/stc-csit.2019.8929774

7. Shynkarenko, V., & Zhevaho, O. (2020). Constructive Modeling of the Software Development Process for Modern Code Review. In 2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT). IEEE. http://doi.org/10.1109/csit49958.2020.9322002

8. Snipes, W., Murphy-Hill, E., Fritz, T., Vakilian, M., Damevski, K., Nair, A. R., & Shepherd, D. (2015). A Practical Guide to Analyzing IDE Usage Data. In The Art and Science of Analyzing Software Data (pp. 85–138). Elsevier. http://doi.org/10.1016/b978-0-12-411519-4.00005-7

9. Edwards, S. H., Snyder, J., Pérez-Quiñones, M. A., Allevato, A., Kim, D., & Tretola, B. (2009). Comparing effective and ineffective behaviors of student programmers. In Proceedings of the fifth international workshop on Computing education research workshop – ICER '09. ACM Press. http://doi.org/10.1145/1584322.1584325

10. Alqadi, B. S., & Maletic, J. I. (2017). An Empirical Study of Debugging Patterns Among Novices Programmers. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. ACM. http://doi.org/10.1145/3017680.3017761

11. Perscheid, M., Siegmund, B., Taeumel, M., & Hirschfeld, R. (2016). Studying the advancement in debugging practice of professional software developers. Software Quality Journal, 25(1), 83–110. http://doi.org/10.1007/s11219-015-9294-2

12. Andreas Zeller (2009). Why Programs Fail, Second Edition: A Guide to Systematic Debugging. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.

13. Ko, A. J., & Myers, B. A. (2008). Debugging reinvented. In Proceedings of the 13th international conference on Software engineering – ICSE '08. ACM Press. http://doi.org/10.1145/1368088.1368130

14. Bryce, R. C., Cooley, A., Hansen, A., & Hayrapetyan, N. (2010). A one year empirical study of student programming bugs. In 2010 IEEE Frontiers in Education Conference (FIE). IEEE. http://doi.org/10.1109/fie.2010.5673143

15. Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2011). XES, XESame, and ProM 6. In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (pp. 60–75). Springer International Publishing. http://doi.org/10.1007/978-3-642-17722-4_5

References

1. LaToza, T. D., & Myers, B. A. (2010). Developers ask reachability questions. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – ICSE'10 (Vol. 1, pp. 185–194). https://doi.org/10.1145/1806799.1806829

2. Shynkarenko, V., & Zhevaho, O. (2020). Development of a toolkit for analyzing software debugging processes using the constructive approach. Eastern-european Journal of Enterprise Technologies, 5(2 (107)), 29–38. https://doi.org/10.15587/1729-4061.2020.215090

3. van der Aalst, W. (2012). Process mining: Overview and opportunities. ACM Transactions on Management Information Systems, 3(2), 1–17. https://doi.org/10.1145/2229156.2229157

4. Rubin, V., Günther, C. W., van der Aalst, W. M. P., Kindler, E., van Dongen, B. F., & Schäfer, W. (2007). Process Mining Framework for Software Processes. In Software Process Dynamics and Agility (pp. 169–181). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-72426-1_15

5. Ardimento, P., Bernardi, M. L., Cimitile, M., & Maggi, F. M. (2019). Evaluating Coding Behavior in Software Development Processes: A Process Mining Approach. In 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). IEEE. http://doi.org/10.1109/icssp.2019.00020

6. Shynkarenko, V., & Zhevago, O. (2019). Visualization of program development process. In 2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT). IEEE. http://doi.org/10.1109/stc-csit.2019.8929774

7. Shynkarenko, V., & Zhevaho, O. (2020). Constructive Modeling of the Software Development Process for Modern Code Review. In 2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT). IEEE. http://doi.org/10.1109/csit49958.2020.9322002

8. Snipes, W., Murphy-Hill, E., Fritz, T., Vakilian, M., Damevski, K., Nair, A. R., & Shepherd, D. (2015). A Practical Guide to Analyzing IDE Usage Data. In The Art and Science of Analyzing Software Data (pp. 85–138). Elsevier. http://doi.org/10.1016/b978-0-12-411519-4.00005-7

9. Edwards, S. H., Snyder, J., Pérez-Quiñones, M. A., Allevato, A., Kim, D., & Tretola, B. (2009). Comparing effective and ineffective behaviors of student programmers. In Proceedings of the fifth international workshop on Computing education research workshop – ICER '09. ACM Press. http://doi.org/10.1145/1584322.1584325

10. Alqadi, B. S., & Maletic, J. I. (2017). An Empirical Study of Debugging Patterns Among Novices Programmers. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. ACM. http://doi.org/10.1145/3017680.3017761

11. Perscheid, M., Siegmund, B., Taeumel, M., & Hirschfeld, R. (2016). Studying the advancement in debugging practice of professional software developers. Software Quality Journal, 25(1), 83–110. http://doi.org/10.1007/s11219-015-9294-2

12. Andreas Zeller (2009). Why Programs Fail, Second Edition: A Guide to Systematic Debugging. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.

13. Ko, A. J., & Myers, B. A. (2008). Debugging reinvented. In Proceedings of the 13th international conference on Software engineering – ICSE '08. ACM Press. http://doi.org/10.1145/1368088.1368130

14. Bryce, R. C., Cooley, A., Hansen, A., & Hayrapetyan, N. (2010). A one year empirical study of student programming bugs. In 2010 IEEE Frontiers in Education Conference (FIE). IEEE. http://doi.org/10.1109/fie.2010.5673143

15. Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2011). XES, XESame, and ProM 6. In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (pp. 60–75). Springer International Publishing. http://doi.org/10.1007/978-3-642-17722-4_5