УДК 004.9
DOI: 10.15587/1729-4061.2020.215090

Разработка средств для анализа процессов отладки программ с применением конструктивного подхода

В. И. Шинкаренко, А. А. Жеваго

Застосовано конструктивно-продукційне моделювання та методи Process Mining у наборі інструментів для моніторингу та аналізу процесу відлагодження. Методи моніторингу процесів розробки і відлагодження є підґрунтям для підвищення рівня практичної підготовки студентів, зменшення часу, який використовуються нераціонально в процесі розробки програм студентом та при контролі процесів виконання завдань викладачем. Процес відлагодження програми розглядається як послідовність дій при роботі з відповідними інструментами. Використовуючи методологію конструктивно-продукційного моделювання, розроблений конструктор для формування журналу відлагоджувальних дій. На основі конструктивної моделі розроблено розширення до інтегрованого середовища розробки (ICP) Microsoft Visual Studio, в якому всі дії по відлагодженню фіксуються в журналах подій. Під час відлагодження у ICP збираються журнали подій, потім виконується перевірка відповідності цих журналів щодо еталонної моделі, для цього використовується ProM (Технічний університет Ейндговена, Нідерланди), платформа для методів Process Mining. Перевіряючи відповідність, можна порівнювати різні процеси виконання і розпізнавати поведінкові схожості і відмінності. Основна мета розробленого інструментарію – зібрати дії по відлагодженню з ICP розробника. Завдяки кращому розумінню того, як студенти розуміють помилки і справляються з ними, можна допомогти новачкам в навчанні програмуванню. Знання про те, як програмісти відлагоджують, можуть спонукати дослідників розробляти більш практично спрямовані методи, викладачів поліпшити свої плани з навчання відлагодженню, а розробників інструментів адаптувати відлагоджувачі до справжніх потреб користувачів. Практично пропонується застосовувати підготовлені інструменти в курсі розробки програмного забезпечення.

Ключові слова: аналіз процесів, відлагодження, конструктивне моделювання, навчання, інженерія програмного забезпечення.

1. Введение

Отладка является одной из наиболее важных, сложных и трудоемких задач в разработке программного обеспечения. Глоссарий IEEE по терминологии разработки программного обеспечения определяет отладку как операцию по обнаружению, локализации и исправлению ошибок в компьютерных программах [1]. Обычно разработчики тратят минимум 30 % времени на отладку и используют интегрированную среду разработки (ИСР), например такую как Microsoft Visual Studio [2].

Улучшение качества кодирования и отладки студентами и начинающими разработчиками является основой обучения программированию. В работах [3, 4] был представлен инструментарий для автоматического мониторинга и визуализации процесса кодирования. Эти подходы предлагается распространить и на процессы отладки.

Формирование навыков эффективной отладки особенно важно для начинающих: они все еще изучают синтаксис и семантику языка программирования, с большой вероятностью создают неправильный код, имеют ограниченные навыки понимания программ и эффективной отладки, что часто приводит к трудностям в понимании и устранении ошибок [5, 6].

Эмпирические исследования по разработке программного обеспечения чаще всего основываются на данных, извлеченных из систем контроля версий и инструментов отслеживания ошибок, но не из ИСР, поскольку они не регистрируют действия разработчиков. Методы Process Mining позволяют анализировать данные, ориентированные на процессы, включая автоматическое формирование моделей процессов и проверку соответствия данных событий эталонной модели [7].

Программирование требует множества компетенций, и их обучение представляет собой центральную проблему в образовании в области информатики и компьютерных наук. В связи с этим актуальными являются исследования, направленные на формирование новых подходов и разработку инструментов для улучшения качества обучения студентов программированию. Студенты не только должны понимать концепции программирования, но и уметь самостоятельно находить решения при столкновении с ошибками. Проверка программ на наличие ошибок, их поиск и исправление это основные компетенции профессиональных разработчиков.

2. Анализ литературных данных и постановка проблемы

Во-первых, рассмотрим подходы к изучению процессов отладки.

Успешная отладка обычно зависит от правильного понимания синтаксиса и семантики программы. Уровень понимания программы и предыдущий опыт работы с подобными ошибками являются двумя ключевыми аспектами, которые отличают новичка от опытного отладчика [8, 9]. Действия по отладке могут значительно улучшить понимание программы, поскольку они требуют от разработчиков читать и понимать код. В [10] опросили разработчиков, чтобы понять, как они исследуют программы. Результат показал, что большую часть своего времени программисты тратят на чтение и понимание исходного кода. Наиболее эффективным средством для понимания кода, по их мнению, является многократный запуск приложения с использованием отладчика. Это подтверждает гипотезу о том, что отладка используется не только для локализации неисправностей, но и для понимания исходного кода.

В [11] разработали лекцию с системой поддержки обучения, чтобы помочь студентам отлаживать с помощью упражнений. Недостаток системы заключается в том, что для выполнения предложенных упражнений требуется уже достаточно высокий уровень навыков и не все испытуемые могут изучить процесс от-

ладки. Кроме того обучение на основе трех упражнений не может показать реальные навыки. Эксперимент по оценке эффективности системы обучения отладке не проводился. В [12] изучали восемь наиболее часто встречаемых ошибок начинающих разработчиков. Написали восемь коротких программ, каждая из которых представляет один тип ошибки, а испытуемым необходимо было сопоставить программу с типом обнаруженной ошибки. В эксперименте принимало участие 59 студентов второго курса Кентского государственного университета. Эксперимент предназначен для отслеживания порядка, в котором участник исправлял ошибки, в дополнение к времени, потраченному на каждую из ошибок. Результаты позволяют установить ошибки, которые являются более сложными. Но при этом не дают возможность оценить уровень навыков отладки студента и не определяют, какие методы и инструменты отладки использовались.

Инструменты для вычислительного мышления, одним из аспектов которого является отладка, были разработаны в [13]. Обучающая игра Light-Bot (Canada) используется для объяснения ее структуры. Недостатком Light-Bot является то, что она используется исключительно для обучения навыкам программирования и отладки, а не их анализа. Кроме того сама имеет очень ограниченный набор инструментов отладки и при переходе к профессиональной ИСР нужно будет проводить процесс обучения повторно. В [14] собрали и проанализировали 450 ошибок, которые ученики не могут решить и по которым они обращаются к учителям за помощью. Для сбора данных разработали вебприложение, где студенты предоставляют краткое описание задания и проблемы, с которой они столкнулись. После чего преподаватель просматривает код вместе со студентом и помогает ему понять проблему. Самые распространенные типы ошибок обсуждаются на занятиях. Главный результат исследования состоит в том, что примерно 22 % проблем связаны с навыками решения проблем, в то время как остальные связаны с комбинацией логических и синтаксических ошибок. Понимание того с какими ошибками сталкиваются студенты позволяет обучать их необходимым навыкам отладки. Преподаватели используют эти данные для постоянного улучшения учебной программы. Информация об эффективности данного подхода не приведена. Недостатком данного подхода является то что студенты должны самостоятельно заходить на сайт и описывать проблемы с которыми они сталкиваются, вместо того чтобы собирать эту информацию напрямую из среды разработки.

В [15], рассмотрели современные тенденции в методах отладки программного обеспечения. Систематическая процедура отладки была предложена в [16]. Исследование показывает, что только несколько студентов применили структурированный подход, но быстро вернулись к неструктурированному. Студентам было предложено найти дефекты в коде, проанализировать их и исправить. Кроме того, студентов попросили задокументировать свой подход. В результате для того, чтобы улучшить навыки отладки у студентов, разработали систематический подход к отладке. Недостатком данного подхода является то, что вывод о навыках делается на основе задокументированного самим студентом подхода и в ручном режиме.

Основываясь на изучении набора данных BlueJ-Blackbox, ошибки, с которыми обычно сталкиваются новички в Java, представлены в [17]. Эмпирические исследования показывают, что семантические ошибки встречаются чаще, чем синтаксические, особенно среди опытных разработчиков. На основе событий компиляции от 250 000 студентов со всего мира анализируется частота и время исправления. Недостатком является то, что анализируются только ошибки компиляции и только из ИСР BlueJ, которая используется для обучения программированию и кардинально отличается от профессиональных сред. В результате после перехода к современным, профессиональным ИСР нужно будет повторно проводить процесс обучения.

В [18] исследовали методы отладки опытных разработчиков программного обеспечения, посредством коротких интервью и наблюдения за каждым из их восьми участников в течение нескольких часов в течение одного рабочего дня. Самым распространенным методом среди опрошенных является «интуитивный». Они формулируют гипотезы о программе, а затем ставят простые эксперименты для их проверки. На основе полученных результатов создана анкета для опроса по отладке. Результаты исследования не подлежат обобщению. Основная проблема – малый масштаб, восемь разработчиков. Еще одна проблема – ограниченный промежуток времени. Практической ценности для обучения студентов результаты исследования не несут. В [19] организовали исследование о том, как программисты ставят точки останова. Анализируя операторы, по которым разработчики устанавливают точки останова, они обнаружили, что 53 % точек останова были установлены в операторах вызова и только 1 % в циклах. Эти данные также подтверждаются исследованием [20]. Вывод этого исследования заключается в том, что установка точек останова и пошаговое выполнение кода являются наиболее часто используемыми методами отладки.

Как видим, многогранные исследования процессов отладки не основываются на моделях процессов и не дают возможности изучения процесса отладки каждого конкретного разработчика ПО.

Во-вторых, рассмотрим, как используют среду программирования при изучении процессов отладки.

Практическое руководство по использованию ИСР приведено в [21]. Основой исследований является идея о том, что инструменты для сбора данных об использовании ИСР обеспечивают более детальное понимание работы разработчиков, чем это было возможно ранее. В работе описаны подходы к извлечению данных из разных сред разработок, частично данные подходы использованы в разработанном расширении. Набор данных, содержащий более 600 часов взаимодействия программиста с ИСР, представлен в [22], из которых более 26 часов сопровождаются видеозаписями экрана компьютера и устными замечаниями разработчиков. Недостатком описанного подхода является то, что логируются и анализируются только события во время написания кода, без его отладки. Кроме того из полученных видеозаписей не извлекается информация, следовательно их анализ может быть только ручным. Как пользователи проводят время, работая в Microsoft Visual Studio, рассмотрено в [23]. Как и в разработанном расширении из ИСР извлекаются данные о процессах разработки и

отладки ПО. Но информация об отладке ограничивается лишь точками останова. Кроме того, извлеченная информация никак не анализируется.

Скрытые марковские модели (СММ) также используются в качестве средства извлечения поведения разработчика из данных взаимодействия с ИСР. Согласно этому подходу [24], изучался ряд отладочных сессий, в которых приняли участие около 200 профессиональных разработчиков из ABB, Inc. Разработанная модель отладки фиксирует поведение разработчиков при установке точек останова, начале отладки и пошаговом выполнении кода. Недостаток предложенного подхода в том, что он собирает данные только о нескольких инструментах отладки и то, что он полуавтоматический. СММ процесса отладки строится вручную экспертом. Кроме того, из ИСР извлекаются только данные процесса отладки, без процесса разработки. В [25] смоделировали пути, выбранные студентами при отладке с использованием СММ. Недостаток работы в том, что описанный подход является только теоретическим и не представлены примеры его даже потенциального использования. Кроме того, процесс отладки рассматривается очень поверхностно, фиксируя лишь запуск программы в режиме отладки, без информации об используемых инструментах.

В [26] реализовали Swarm Debug Infrastructure (SDI), которая предоставляет инструменты для сбора, обмена и извлечения отладочных действий. Программисты могут использовать общий опыт предыдущих сеансов отладки. SDI оценивался в эмпирическом эксперименте с 10 инженерами. SDI предназначен лишь для сбора и обмена информацией про установленные точки останова и пути отладки. И не осуществляет анализ процесса отладки и навыков разработчиков. В [27] представляют онлайн-инструмент Ladebug (Digital Equipment Corporation, USA), разработанный для поддержки обучения навыкам отладки. В этом инструменте учащиеся используют систематический процесс отладки для выявления и исправления ошибок в заданных упражнениях. Представленный инструмент используется лишь для обучения основам отладки и не анализирует навыки студентов. Кроме того, процесс отладки проходит внутри инструмента, который используется лишь для обучения с очень ограниченным набором инструментов. При переходе в профессиональные ИСР студентам придется заново проходить процесс обучения уже реальным, а не учебным инструментам. Когнитивные процессы учащихся при отладке приложений, использующих отслеживание глаз, изучались в [28]. Движение глаз учащихся регистрировалось, чтобы узнать, как ведут себя ученики с высокой и низкой эффективностью во время отладки. Исследование показало, что новички в программировании следовали линейному построчному подходу при отладке компьютерных программ, в то время как студенты с предшествующим опытом программирования следовали более логичному и стратегическому подходу. Поэтому решили разделить студентов на группы в соответствии с их успехами, создав среду, в которой они могут думать вслух. Недостаток такого подхода заключается в том, что невозможно анализировать процесс отладки во время индивидуальной работы.

И наконец, в-третьих, рассмотрим, как применяются средства Process Mining.

За последнее десятилетие применение Process Mining показало эффективность анализа процессов на основе данных о событиях. Целью Process Mining является выявление, мониторинг и улучшение процессов, применяя данные из журнала событий, полученные из информационной системы [29]. Целевая группа IEEE выпустила Process Mining Manifesto [30]. Этот манифест был поддержан 53 организациями и 77 экспертами Process Mining. Он направлен на продвижение Process Mining. Кроме того, определяя набор правил и перечисляя критические проблемы, этот манифест должен служить руководством для разработчиков программного обеспечения, ученых и конечных пользователей.

Process Mining может в равной степени применяться к программному обеспечению [31]. Используя методы Process Mining, в [32] изучали, как программисты взаимодействуют с репозиториями программного обеспечения. В [33] исследовали записи в системах отслеживания проблем. Общий недостаток этих работ в том, что данные извлекаются из систем, которые не позволяют оценить вклад студента в конечный результат, потому что не предоставляют информацию о процессе написания и отладки программного обеспечения. В [34] использовали проверку соответствия для проверки поведения разработчиков. Оценили действия, выполненные 40 начинающими разработчиками, выполняющими действия по кодированию в пяти сессиях разработки. В работе в основном фокусируются на использовании подхода, основанного на проверке соответствия, сравнении выполнения процессов, записанных в журналах событий, с некоторым ожидаемым поведением, представленным в виде модели процесса. Недостаток предложенного подхода заключается в том, что процесс разработки и отладки рассматривается как один процесс. Подобный подход был представлен в [35], но там представлена только базовая концепция, оставляя его реализацию и проверку для будущей работы.

Систематизация результатов исследования позволяет утверждать о недостаточном знании того, как программисты устраняют проблемы в процессе отладки программного обеспечения. Все это позволяет утверждать, что целесообразным является проведение исследования, посвященного разработке средств отслеживания, моделирования и анализа процессов отладки программного обеспечения

3. Цель и задачи исследования

Целью исследования является разработка средств мониторинга и анализа процессов отладки программного обеспечения. Анализируя то, как разработчики используют ИСР, можно обнаружить шаблоны поведения программистов во время отладки и выявить проблемы, с которыми они сталкиваются.

Для достижения цели были поставлены следующие задачи:

– разработать конструктор для формирования журнала событий, отображающий процесс отладки;

– разработать средства для фиксации отладочных событий из ИСР разработчика;

– сформировать модель процесса отладки с помощью методов Process Mining.

4. Конструирование журнала событий с данными об использовании ИСР во время отладки

Для формализации процесса сбора данных об использовании ИСР во время отладки, применено конструктивно-продукционное моделирование (КПМ). Основы КПМ приведены в [37–40]. КПМ может применяться для моделирования и формализации любых конструкций и конструктивных процессов. Инструменты КПМ применялись для решения таких задач, как:

– создание расписания университетского курса [41];

– моделирование вспышек молнии во фронте грозы [42];

– представление геометрических фракталов [43];

– моделирование адаптации алгоритмов сжатия [44];

– формализации и автоматизации процесса сравнение документов для выявления текстовых заимствований [45] и многих других.

Широкий спектр этих задач демонстрирует универсальность и перспективы использования КПМ для решения проблем различных предметных областей. Эти работы раскрывают универсальность и высокую общность этого метода моделирования.

Первым этапом проектирования является специализация обобщенного конструктора [37]. Специализация определяет семантическую природу носителя, цель конструирования, конечный набор операций, их семантику и атрибуты, порядок выполнения и ограничения [37, 38].

В неформальном виде онтология обобщенного конструктора представлена в [37, 38]. Основные положения онтологического сопровождения КПМ представлены в [46, 47]. В работе представлены только те составляющие, которые необходимы для дальнейшего изложения.

Целью конструктора является создание журнала событий, отображающий процесс отладки.

Начальные условия – нетерминал σ, с которого начинается вывод.

Условия завершения – все события обработаны.

Специализация обобщенного конструктора:

$$C = \langle \mathrm{M}, \Sigma, \Lambda \rangle \,{}_{S}\!\mapsto \mathrm{C}_{L} = \langle \mathrm{M}_{L}, \Sigma_{L}, \Lambda_{L} \rangle, \tag{1}$$

где ${}_{S}\!\mapsto$ – операция специализации (выполняемая внешним исполнителем), M_L – неоднородный пополняемый носитель, включающий множество терминалов и нетерминалов, $\sum_L$ – сигнатура отношений и соответствующих операций, Λ_L – информационное обеспечение конструирования.

Терминалы и их атрибуты:

– ${}_{id,\,sdt,\,fdt,\,ss,\,el,\,d,\,p}S$ – сеанс отладки, *id* – её идентификатор, *sdt*, *fdt* – время начала и конца, *ss* – массив снимков (снимок – файл кода программы в некоторый момент времени) в начале сеанса, *el* – массив событий во время сеанса отладки, *d* – информация о разработчике, *p* – информация о проекте;

– атрибуты *el* $\left({}_{e\,index,\,l}el\right)$: *index* – индекс события *e* в массиве, *l* – размер массива;

– $_{id,\ t,\ cdt,\ fp,\ ln,\ context}e$ – событие во время сеанса отладки, *id* – идентификатор, *t* – тип, *cdt* – время появления события, *fp* – путь к отлаживаемому файлу, *ln* – строка кода с которой связано событие, *context* – контекст события, объект динамической структуры с информацией об окружении события (ИСР, проекте, разработчике, файле), для разных событий структура объекта может быть разной;

– атрибуты *d* ($_{id,\ name}d$): *id* – идентификатор, *name* – имя разработчика;

– атрибуты *p* ($_{id,\ name}p$): *id* – идентификатор, *name* – название проекта;

– $_{context}sdse$ – событие запуска отладки от внешнего исполнителя (ИСР);

– $_{context}de$ – событие отладки от ИСР;

– $_{context}edse$ – событие завершения отладки от ИСР;

– $_{traces}log$ – файл журнала событий в XES формате, *traces* – массив последовательностей (событий) с атрибутами $\left(_{trace}index,_{l}traces\right)$: *index* – индекс последовательности *trace*, *l* – размер массива *traces*;

– $_{id,\ sdt,\ fdt,\ p,\ d,\ events}trace$ – массив событий, при однократном выполнении процесса, *events* – массив событий во время сеанса отладки.

Конструктор имеет следующие операции над атрибутами

– $\circ(t)$ – присвоение значения терминалу *t* внешним исполнителем;

– $\prec(sdse, s)$ – создание сеанса отладки *s* по событию *sdse*;

– $\succ(s, de)$ – добавление события *de* к сеансу отладки *s*;

– $\cong(s, edse)$ – завершение сеанса отладки *s* по событию *edse*;

– $\triangleleft(did, pid, log)$ – создание файла журнала событий *log* для проекта *pid* и разработчика *did*;

– $\triangleright(de, s, log)$ – перемещение события *de* с сеанса отладки *s* в файл журнала событий *log*;

– $\approx(log)$ – сохранение файла журнала событий.

Сигнатура $\Sigma_L = \langle \Xi, \Theta, \Phi, \{\rightarrow, \hookleftarrow\}, \Psi \rangle$ содержит множество операций и отношений, где $\Xi \supset \{\bullet, :\}$ – операции связывания и преобразования элементов носителя, $\Theta = \{\Rightarrow, |\Rightarrow, ||\Rightarrow\}$ – операции подстановки и вывода, Φ – операции над атрибутами, а также отношения подстановки ($\rightarrow$) и атрибутивности ($\hookleftarrow$), $\Psi = \{\psi_i : \langle s_i, g \rangle_i\}$ – множество правил подстановки, s_i – последовательность отношений подстановки, g_i – последовательность операций над атрибутами.

Интерпретация заключается в связывании алгоритмов, реализующих определенную алгоритмическую структуру (конструктор алгоритмов), с операциями сигнатуры. В процессе интерпретации связываются модели конструктора и внутреннего исполнителя. В результате получается конструктивная система, способная и имеющая средства для конструирования [37, 38].

Для интерпретации C_L необходимо уточнить базовую алгоритмическую структуру (БАС) [37, 38].

Пусть имеется следующая БАС:

$$C_{A,L} = \langle \mathrm{M}_{A,L}, V_{A,L}, \Sigma_{A,L}, \Lambda_{A,L} \rangle, \qquad (2)$$

где $V_{A,L}$ – конечное множество базовых алгоритмов $A_i|_{X_i}^{Y_i}$ внутреннего исполнителя конструирования с множествами входных и выходных данных X_i и Y_i.

Следующие алгоритмы реализуют операции над атрибутами:

$$A_1|_t^t,\ A_2|_{s,sdse}^s,\ A_3|_{s,de}^s,\ A_4|_{s,edse}^s,$$

$$A_5|_{did,pid,\log}^{\log},\ A_6|_{de,s,\log}^{\log},\ A_7|_{\log}^{\log}. \tag{3}$$

Создадим конструктивную систему:

$$\left\langle \begin{array}{l} C_L = \langle M_L, \Sigma_L, \Lambda_L \rangle, \\ C_{A,L} = \langle M_{A,L}, V_{A,L}, \Sigma_{A,L}, \Lambda_{A,L} \rangle \end{array} \right\rangle,$$

$$_I\mapsto C_{I,L} = \langle M_{I,L}, \Sigma_{I,L}, \Lambda_{I,L} \rangle,$$

$$\Lambda_{I,L} = \Lambda_L \cup \left\{ \begin{array}{l} \left(A_1|_t^t \hookleftarrow \circ\right), \left(A_2|_{s,sdse}^s \hookleftarrow \prec\right), \left(A_3|_{s,de}^s \hookleftarrow \succ\right), \\ \left(A_4|_{s,edse}^s \hookleftarrow \cong\right), \left(A_5|_{did,pid,\log}^{\log} \hookleftarrow \lhd\right), \\ \left(A_6|_{de,s,\log}^{\log} \hookleftarrow \rhd\right), \left(A_7|_{\log}^{\log} \hookleftarrow \approx\right) \end{array} \right\}, \tag{4}$$

где $_I\mapsto$ – операция интерпретации.

Конкретизация конструктора заключается в определении конкретных правил подстановки, ограничений, начальных условий и условий завершения конструирования, базисных элементов носителя с их свойствами и значениями свойств. После операций интерпретации и конкретизации, выполненных внешним исполнителем, в конструктивной системе есть все необходимое для автономного создания конструкций [37–40].

Выполним конкретизацию конструктора C_L для создания журнала отладочных действий:

$$C_{I,L} = \langle M_{I,L}, \Sigma_{I,L}, \Lambda_{I,L} \rangle_K \mapsto C_L = \langle M_{K,L}, \Sigma_{K,L}, \Lambda_{K,L} \rangle\rangle, \tag{5}$$

где $_K\mapsto$ – операция конкретизации.

Ниже представлены правила подстановки (6)–(9).

Правило s_1 применяется, если событие *sdse* получено от внешнего исполнителя. В результате выполнения правила создается сессия отладки *s*

$$s_1 = \langle \sigma \to (sdse \bullet \alpha) \rangle,\ \ g_1 = \langle \circ(sdse), \prec(sdse, s) \rangle. \tag{6}$$

Правило s_2 применяется, если получено событие *de*. В результате событие *de* добавляется к сессии отладки *s*

$$s_2 = \langle \alpha \to de \bullet \alpha \rangle, \quad g_2 = \langle \circ(de), \succ(s, de) \rangle. \tag{7}$$

Правило s_3 применяется, если получено событие *edse*. В результате закрывается сессия отладки *s*

$$s_3 = \langle \alpha \to edse \bullet \beta \rangle, \quad g_3 = \langle \circ(edse), \cong(s, edse) \rangle. \tag{8}$$

В результате выполнения правила s_4, создается файл журнала событий *log*. Файл будет заполнен событиями из предыдущих сеансов отладки разработчика для этого проекта или пуст и заполняется событиями из текущего сеанса отладки

$$s_4 = \langle \beta \to log \rangle,$$

$$g_4 = \langle \triangleleft(id \hookleftarrow d \hookleftarrow s, id \hookleftarrow p \hookleftarrow s, log), \triangleright(de, s, log) \rangle. \tag{9}$$

Реализация, выполняемая внутренним исполнителем системы, состоит из формирования конструкции из элементов-носителей путем выполнения алгоритмов, связанных с операциями подстановки. Только тот конструктор, который был до этого специализирован, интерпретирован и конкретизирован, может быть реализован. В результате формируется файл журнала отладки.

5. Разработка средств для фиксации отладочных событий из ИСР разработчика

На основе конструктивной модели разработано расширение для Microsoft Visual Studio, в котором все действия по отладке фиксируются в журналах событий. Для хранения и контроля отслеживания событий создана модель данных, которая соответствует ранее представленным терминалам с их атрибутами. На рис. 1 показана предложенная модель в виде диаграммы классов.

Диаграмма классов основана на событийной модели процесса отладки. События возникают, если программист выполняет некоторые действия во время сеанса отладки. Они содержат тип, время, контекст и путь к файлу отладки с номером строки, на которой произошло событие, если это необходимо. Мы храним не только простую информацию о выполненных командах, но и конкретные данные (объект динамической структуры с информацией об окружении события, контекст) в зависимости от типа события. *Developer* – пользователь, который запускает и выполняет сеансы отладки. *Project* – решение из набора компонентов. *Session* представляет собой сеанс отладки. Он содержит информацию о разработчике, проекте и события отладки. Каждое событие связано с конкретным типом. Каждый тип события представлен соответствующим классом, который является реа-

лизацией абстрактного базового класса *Event*. Перечисление *EventType* содержит все возможные типы событий, которые отслеживаются.

На рис. 2 представлена архитектура, верхнего уровня, разработанного программного средства.

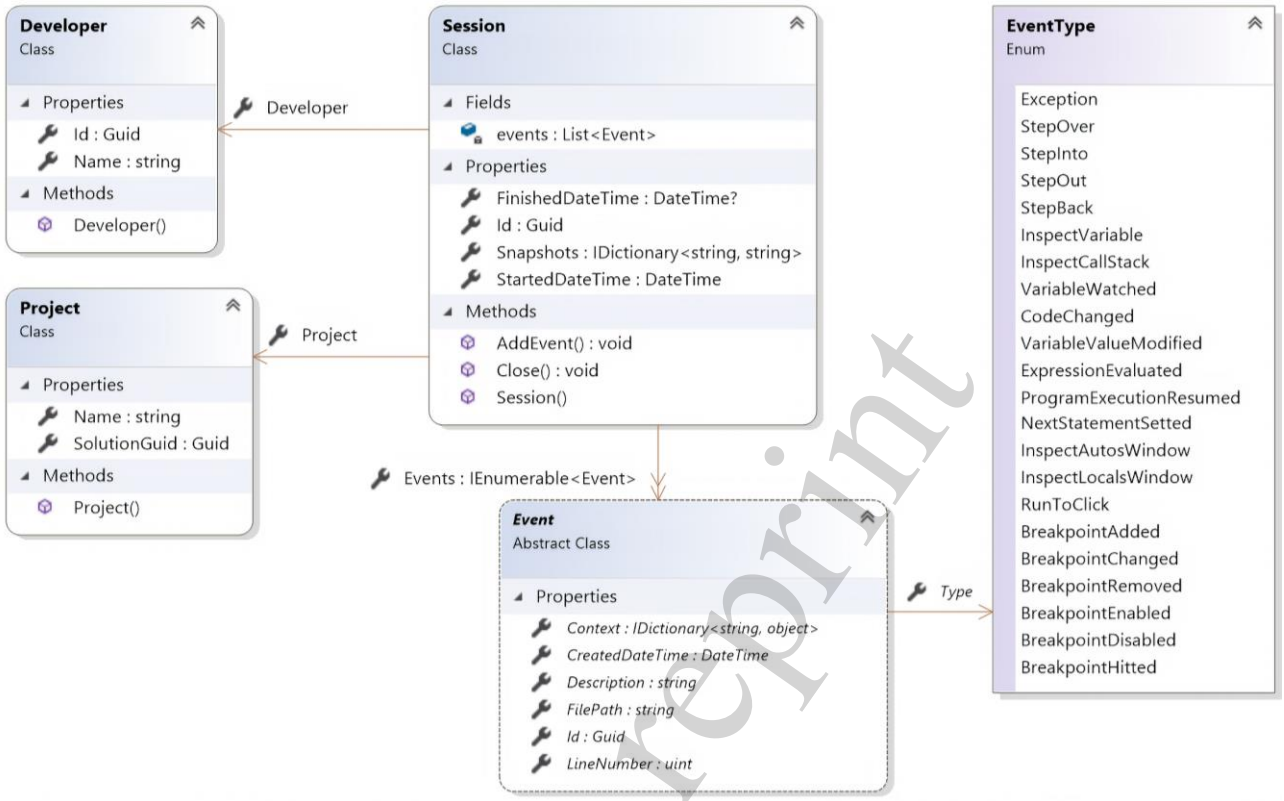

Рис. 1. Диаграмма классов модели данных для хранения и управления отслеживанием событий

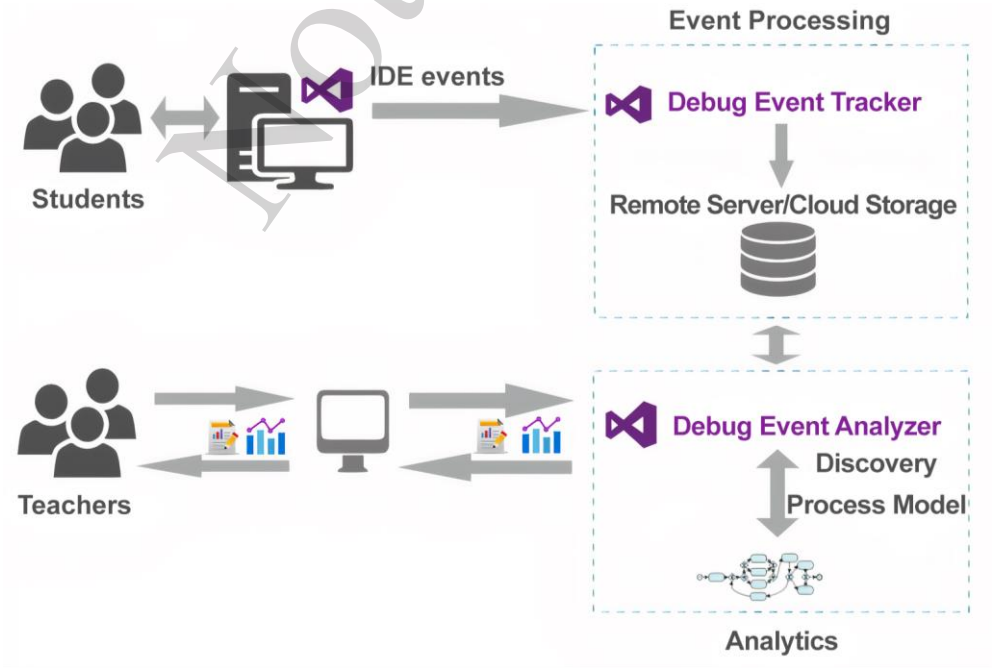

Рис. 2. Архитектура средства для мониторинга и анализа процесса отладки

На рис. 2 представлены два основных компонента: Debug Event Tracker и Debug Event Analyzer. Debug Event Tracker – это расширение к Microsoft Visual Studio, написанное на C#. Компонент отвечает за сбор данных сессий отладки текущего проекта и отправку их на удаленный сервер. Расширение не только фиксирует, какие события происходят в ИСР, но также хранит соответствующую контекстную информацию о них. Debug Event Analyzer отвечает за анализ операций отладки с использованием методов Process Mining.

6. Обработка журнала событий методами Process Mining

Процесс Mining предназначен для формирования моделей, проверки и улучшения реальных процессов путем извлечения информации из журналов событий, которые обеспечивают понимание процесса. Результат Process Mining может варьироваться от полной модели процесса до описания наиболее частых путей процесса или отклонений.

Отправной точкой Process Mining является журнал событий. Каждое событие в таком журнале относится к действию, которое может быть выполнено на ресурсе в определенное время и для конкретного сеанса. Журнал событий обычно структурируется как набор трасс, где каждая трасса составляет цепочку действий, созданных в результате одного выполнения процесса (сеанса). Как минимум, запись события включает в себя идентификатор сеанса процесса, к которому применяется событие, время и ряд дополнительных атрибутов. Описание атрибутов журнала событий приведено в табл. 1.

Журнал событий формируется в формате eXtensible Event Stream (XES) [36], который является стандартным форматом для Process Mining, разработанным рабочей группой IEEE для регистрации событий.

Таблица 1
Атрибуты событий

Атрибут	Уровень	Описание
Name	Trace	Идентификатор сеанса отладки
StartedDateTime	Trace	Время начала сеанса отладки
FinishedDateTime	Trace	Время окончания сеанса отладки
Project	Trace	Идентификатор проекта
Developer	Trace	Идентификатор разработчика
Activity	Event	Наименование события
Timestamp	Event	Время появления события
Context	Event	Контекст события
Resource	Event	Путь к отлаживаемому файлу
LineNumber	Event	Номер строки, с которой связано событие

Первый метод Process Mining – формирование. Метод формирования принимает журнал событий, состоящий из записи всех действий, которые происходят в процессе отладки программного обеспечения, и формирует модель, которая представляет, как и в каком порядке, процесс был выполнен. Для формиро-

вания модели процесса отладки из журнала событий, используется ProM. ProM – стандартная платформа с открытым исходным кодом, де-факто стандарт, для реализации Process Mining [48].

В ProM импортировали журналы событий пяти сеансов отладки в формате XES, отслеживаемые при использовании ИСР во время процесса отладки программного обеспечения, чтобы подробно показать, как выполнялся процесс отладки. На рис. 3 – представление трассировки, где каждая трасса представляет один сеанс отладки.

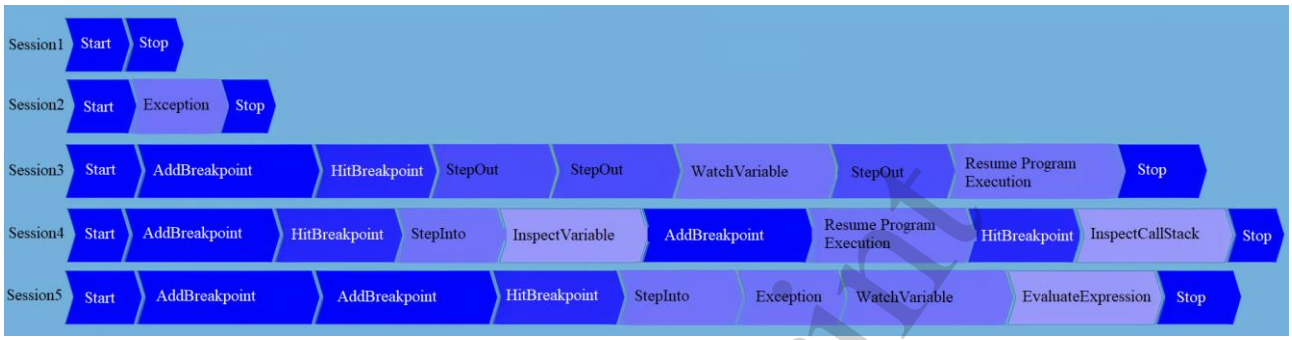

Рис. 3. Трассировка

Ниже представлены конструкции представления процесса отладки, обеспечивающие взаимнооднозначное соответствие (представления и процесса) для первых двух трасс (10)–(11).

$$\sigma \overset{s_1}{\Rightarrow} sdse \bullet \alpha \overset{s_3}{\Rightarrow} sdse \bullet edse \bullet \beta \overset{s_4}{\Rightarrow} sdse \bullet edse \bullet log, \quad (10)$$

$$\sigma \overset{s_1}{\Rightarrow} sdse \bullet \alpha \overset{s_2}{\Rightarrow} sdse \bullet de \bullet \alpha \overset{s_3}{\Rightarrow} sdse \bullet de \bullet edse \bullet \beta \overset{s_4}{\Rightarrow} sdse \bullet de \bullet edse \bullet log. \quad (11)$$

ProM's Inductive Visual Miner [49], который предоставляет графы прямого следования (ГПС), был использован для формирования модели процесса отладки. ГПС представляет действия в форме прямоугольников и связи действий, если одно из них непосредственно следует за другим. Кроме того, каждое ребро имеет вес, указывающий количество записей в журнале событий. Результатом этого этапа является модель процесса, показанная на рис. 4.

Второй метод Process Mining – проверка соответствия. Посредством проверки соответствия можно сопоставлять различные реализации процессов и выяснять поведенческие сходства и различия. В исследовании проверка соответствия проводится дважды: при воспроизведении обнаруженной модели на журнал событий и при измерении отклонения между предопределенной моделью и обнаруженной моделью. Фитнес функция [30] используется для проверки соответствия. Модель имеет отличную пригодность, если все трассы могут быть воспроизведены моделью от начала до конца. Фитнес характеризуется числом от 0 (очень плохо) до 1 (отлично).

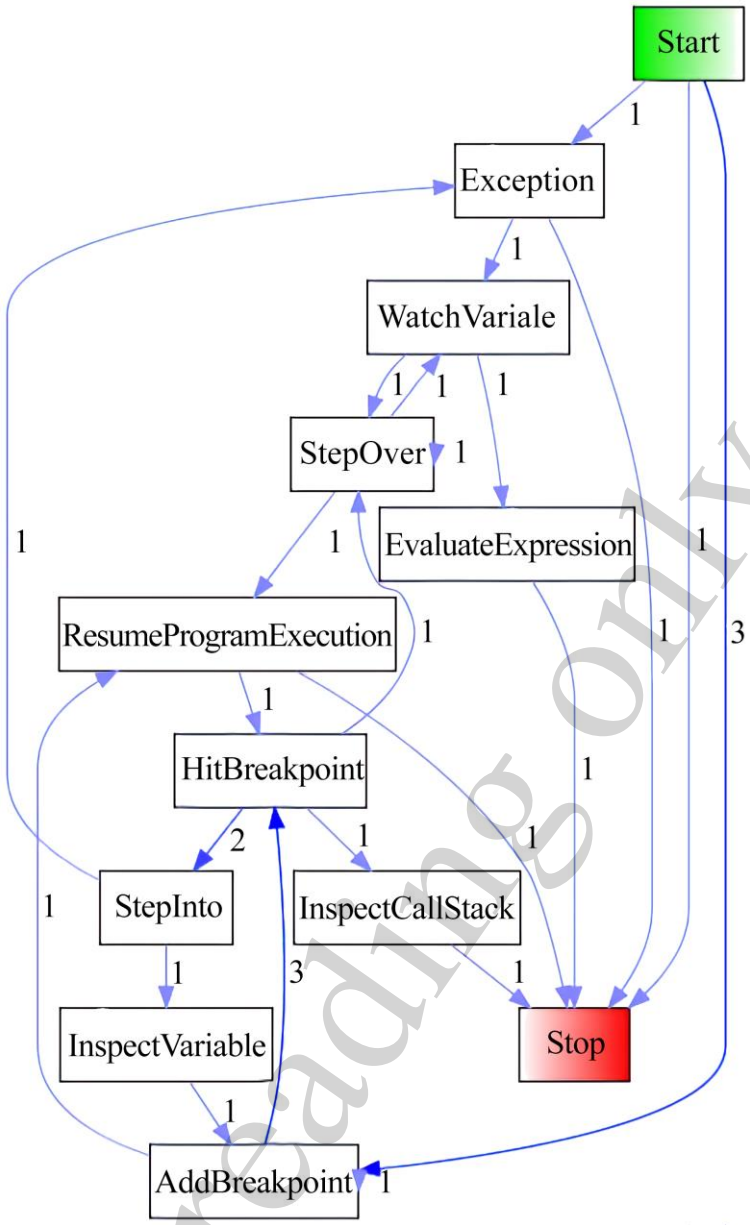

Рис. 4. Модель процесса отладки сформирована с использованием Inductive Visual Miner

Предопределенная модель показывает, какие переходы между действиями возможны (рис. 5).

Набор инструментов ProM содержит плагины, которые позволяют рассчитать пригодность по модели и файлу журнала событий, а также пригодность по двум моделям. Проверка соответствия выполняется с использованием плагина Visualize deviations ProM [49]. Пригодность сформированного с помощью расширения, к Visual Studio, файла логов по отношению к полученной модели равно единице, что подтверждает достоверность и адекватность полученной модели.

Рис. 5. Допустимые переходы

7. Обсуждение результатов исследования разработанных средств анализа процессов отладки программ

Разработаны три составляющие исследования:

– плагин к Visual Studio, который позволяет получить информацию о событиях при кодировании и отладке программ. Среда Visual Studio предоставляет возможность получение информации о действиях программиста с привязкой к тексту программы. Функциональность плагина, обеспечивающая сбор информации о процессе отладки, представлена на рис. 2.

– расширение функциональности плагина, связанной с привязкой событий к программным сущностям, стало возможным, основываясь на формализме конструктора (1)–(9).

– средства и методы Process Mining (именно ProM) для отслеживания и анализа процессов. Формирование модели конкретного процесса представлено на рис. 3, 4. Анализ выполняется по обобщенной модели процесса как показано на рис. 5.

Для совершенствования процессов отладки и обучению отладке используется несколько подходов [8–28]. Только часть из них [21–28] основывается на объективной информации, получаемой от инструментария разработки и отладки программ. Анализ процесса осуществляется вручную. В рамках данного исследования удалось автоматизировать анализ процесса отладки, применив известные средства и методы Process Mining. Проверка соответствия выполняется с использованием плагина Visualize deviations ProM.

Представлены результаты исследования, в котором проанализированы журналы событий пяти сеансов отладки с использованием инструмента Process Mining. После применения методов Process Mining к журналам событий получена модель процесса отладки программного обеспечения. Результаты показывают, что используемые методы Process Mining полезны для понимания того, как программисты выполняют действия по отладке и с какими трудностями они обычно сталкиваются.

Таким образом, развивая предложенные в [21–28] средства и методы сбора информации о процессах отладки и, автоматизируя формирование, проверку и анализ моделей процесса, разработан полный технологический цикл оценки качества процесса отладки. Данное исследование позволяет как преподавателю, так и обучаемому программисту, получить дополнительную информацию о процессе отладки и его качестве, тем самым, расширяя возможности по эффективному управлению процессом отладки, совершенствованию навыков. Как известно, чем больше информации известно о процессе, тем эффективнее может быть управление.

Результаты показывают, что используемые методы Process Mining полезны для понимания того, как программисты выполняют действия по отладке и с какими трудностями они обычно сталкиваются.

Основываясь на результатах исследования, адаптивные методы обучения могут быть применены для студентов с различной степенью академической успеваемости.

Естественно, предложенный подход не является универсальным. Пока что он применим только к конкретной среде разработки – Visual Studio. Для других сред разработки необходимо доработать только плагин.

Эта статья – первый шаг к пониманию навыков отладки. Конечная цель – улучшить навыки отладки разработчиков. В будущем предполагается разработка рекомендательных системы для преподавателя и обучающегося процессам отладки программ.

8. Выводы

1. Для формализации процесса сбора данных об использовании ИСР во время отладки используется подход конструктивно-продукционного моделиро-

вания. Разработан конструктор, целью которого является создание файла журнала событий отладочной деятельности в формате XES.

2. На основе конструктивной модели разработано расширение для Microsoft Visual Studio, в котором все действия по отладке фиксируются в журналах событий. Анализ взаимодействия между программистами и ИСР помогает исследователям интерпретировать поведение разработчиков.

3. Применены методы Process Mining для построения модели процесса отладки и ее проверки. Эксперимент проводился с использованием платформы с открытым исходным кодом ProM. Результаты показали, что процесс отладки программного обеспечения может быть эффективно анализирован с использованием подхода Process Mining.

Литература

1. IEEE Standard Glossary of Software Engineering Terminology (1990). doi: https://doi.org/10.1109/ieeestd.1990.101064
2. LaToza, T. D., Myers, B. A. (2010). Developers ask reachability questions. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE ’10. doi: https://doi.org/10.1145/1806799.1806829
3. Shynkarenko, V., Zhevago, O. (2019). Visualization of program development process. 2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT). doi: https://doi.org/10.1109/stc-csit.2019.8929774
4. Shynkarenko, V., Zhevaho, O. (2020). Constructive modeling of the software development process for modern code review. In IEEE 2020 15th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT 2020.
5. Denny, P., Luxton-Reilly, A., Tempero, E., Hendrickx, J. (2011). Understanding the syntax barrier for novices. Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE ’11. doi: https://doi.org/10.1145/1999747.1999807
6. Denny, P., Luxton-Reilly, A., Carpenter, D. (2014). Enhancing syntax error messages appears ineffectual. Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education - ITiCSE ’14. doi: https://doi.org/10.1145/2591708.2591748
7. Pegoraro, M., van der Aalst, W. M. P. (2019). Mining Uncertain Event Data in Process Mining. 2019 International Conference on Process Mining (ICPM). doi: https://doi.org/10.1109/icpm.2019.00023
8. Bers, M. U., Flannery, L., Kazakoff, E. R., Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Computers & Education, 72, 145–157. doi: https://doi.org/10.1016/j.compedu.2013.10.020
9. Lee, G. C., Wu, J. C. (1999). Debug It: A debugging practicing system. Computers & Education, 32 (2), 165–179. doi: https://doi.org/10.1016/s0360-1315(98)00063-3

10. Maalej, W., Tiarks, R., Roehm, T., Koschke, R. (2014). On the Comprehension of Program Comprehension. ACM Transactions on Software Engineering and Methodology, 23 (4), 1–37. doi: https://doi.org/10.1145/2622669

11. Yamamoto, R., Noguchi, Y., Kogure, S., Yamashita, K., Konishi, T., Itoh, Y. (2016). Design of a learning support system and lecture to teach systematic debugging to novice programmers. In ICCE 2016 – 24th International Conference on Computers in Education: Think Global Act Local – Main Conference Proceedings, 276–281.

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

13. Gouws, L. A., Bradshaw, K., Wentworth, P. (2013). Computational thinking in educational activities. Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '13. doi: https://doi.org/10.1145/2462476.2466518

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

15. Ghosh, D., Singh, J. (2019). A Systematic Review on Program Debugging Techniques. Smart Computing Paradigms: New Progresses and Challenges, 193–199. doi: https://doi.org/10.1007/978-981-13-9680-9_16

16. Bottcher, A., Thurner, V., Schlierkamp, K., Zehetmeier, D. (2016). Debugging students' debugging process. 2016 IEEE Frontiers in Education Conference (FIE). doi: https://doi.org/10.1109/fie.2016.7757447

17. Altadmri, A., Brown, N. C. C. (2015). 37 Million Compilations. Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15. doi: https://doi.org/10.1145/2676723.2677258

18. 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. doi: https://doi.org/10.1007/s11219-015-9294-2

19. Petrillo, F., Mandian, H., Yamashita, A., Khomh, F., Gueheneuc, Y.-G. (2017). How Do Developers Toggle Breakpoints? Observational Studies. 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). doi: https://doi.org/10.1109/qrs.2017.39

20. Beller, M., Spruit, N., Spinellis, D., Zaidman, A. (2018). On the dichotomy of debugging behavior among programmers. Proceedings of the 40th International Conference on Software Engineering. doi: https://doi.org/10.1145/3180155.3180175

21. 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. The Art and Science of Analyzing Software Data, 85–138. doi: https://doi.org/10.1016/b978-0-12-411519-4.00005-7

22. Yamashita, A., Petrillo, F., Khomh, F., Guéhéneuc, Y.-G. (2018). Developer interaction traces backed by IDE screen recordings from think aloud sessions.

Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18. doi: https://doi.org/10.1145/3196398.3196457

23. Bellman, C., Seet, A., Baysal, O. (2018). Studying developer build issues and debugger usage via timeline analysis in visual studio IDE. Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18. doi: https://doi.org/10.1145/3196398.3196463

24. Damevski, K., Chen, H., Shepherd, D., Pollock, L. (2016). Interactive exploration of developer interaction traces using a hidden Markov model. Proceedings of the 13th International Workshop on Mining Software Repositories - MSR ’16. doi: https://doi.org/10.1145/2901739.2901741

25. Piech, C., Sahami, M., Koller, D., Cooper, S., Blikstein, P. (2012). Modeling how students learn to program. Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE ’12. doi: https://doi.org/10.1145/2157136.2157182

26. Petrillo, F., Soh, Z., Khomh, F., Pimenta, M., Freitas, C., Gueheneuc, Y.-G. (2016). Understanding interactive debugging with Swarm Debug Infrastructure. 2016 IEEE 24th International Conference on Program Comprehension (ICPC). doi: https://doi.org/10.1109/icpc.2016.7503740

27. Luxton-Reilly, A., McMillan, E., Stevenson, E., Tempero, E., Denny, P. (2018). Ladebug: an online tool to help novice programmers improve their debugging skills. Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018. doi: https://doi.org/10.1145/3197091.3197098

28. Lin, Y.-T., Wu, C.-C., Hou, T.-Y., Lin, Y.-C., Yang, F.-Y., Chang, C.-H. (2016). Tracking Students’ Cognitive Processes During Program Debugging – An Eye-Movement Approach. IEEE Transactions on Education, 59 (3), 175–186. doi: https://doi.org/10.1109/te.2015.2487341

29. Van der Aalst, W. (2012). Process Mining. ACM Transactions on Management Information Systems, 3 (2), 1–17. doi: https://doi.org/10.1145/2229156.2229157

30. Van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T. et. al. (2012). Process Mining Manifesto. Lecture Notes in Business Information Processing, 169–194. doi: https://doi.org/10.1007/978-3-642-28108-2_19

31. Rubin, V. A., Mitsyuk, A. A., Lomazova, I. A., van der Aalst, W. M. P. (2014). Process mining can be applied to software too! Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM ’14. doi: https://doi.org/10.1145/2652524.2652583

32. Poncin, W., Serebrenik, A., Brand, M. van den. (2011). Process Mining Software Repositories. 2011 15th European Conference on Software Maintenance and Reengineering. doi: https://doi.org/10.1109/csmr.2011.5

33. Sebu, M. L., Ciocarlie, H. (2014). Applied process mining in software development. 2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI). doi: https://doi.org/10.1109/saci.2014.6840098

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

35. Caldeira, J., Abreu, F. B. e. (2016). Software Development Process Mining: Discovery, Conformance Checking and Enhancement. 2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC). doi: https://doi.org/10.1109/quatic.2016.061

36. 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. Lecture Notes in Computer Science, 60–75. doi: https://doi.org/10.1007/978-3-642-17722-4_5

37. Shynkarenko, V. I., Ilman, V. M. (2014). Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure. Cybernetics and Systems Analysis, 50 (5), 655–662. doi: https://doi.org/10.1007/s10559-014-9655-z

38. Shynkarenko, V. I., Ilman, V. M. (2014). Constructive-Synthesizing Structures and Their Grammatical Interpretations. II. Refining Transformations. Cybernetics and Systems Analysis, 50 (6), 829–841. doi: https://doi.org/10.1007/s10559-014-9674-9

39. Shynkarenko, V. I., Ilman, V. M., Skalozub, V. V. (2009). Structural models of algorithms in problems of applied programming. I. Formal algorithmic structures. Cybernetics and Systems Analysis, 45 (3), 329–339. doi: https://doi.org/10.1007/s10559-009-9118-0

40. Shynkarenko, V. I., Ilman, V. M., Skalozub, V. V. (2009). Structural models of algorithms in problems of applied programming. II. Structural-algorithmic approach to software simulation. Cybernetics and Systems Analysis, 45 (4), 544–550. doi: https://doi.org/10.1007/s10559-009-9122-4

41. Shinkarenko, V. I., Zhevago, O. O. (2019). Generating university course timetable using constructive modeling. Radio Electronics, Computer Science, Control, 3, 152–162. doi: https://doi.org/10.15588/1607-3274-2019-3-17

42. Shynkarenko, V., Lytvynenko, K., Chyhir, R., Nikitina, I. (2019). Modeling of Lightning Flashes in Thunderstorm Front by Constructive Production of Fractal Time Series. Advances in Intelligent Systems and Computing, 173–185. doi: https://doi.org/10.1007/978-3-030-33695-0_13

43. Shynkarenko, V. I. (2019). Constructive-Synthesizing Representation of Geometric Fractals. Cybernetics and Systems Analysis, 55 (2), 186–199. doi: https://doi.org/10.1007/s10559-019-00123-w

44. Shynkarenko, V. I., Vasetska, T. M. (2015). Modeling the Adaptation of Compression Algorithms by Means of Constructive-Synthesizing Structures. Cybernetics and Systems Analysis, 51 (6), 849–862. doi: https://doi.org/10.1007/s10559-015-9778-x

45. Kuropiatnyk, O., Shynkarenko, V. (2020). Text borrowings detection system for natural language structured digital documents. In CEUR Workshop Proceedings, 2604, 294–305.

46. Skalozub, V., Ilman, V., Shynkarenko, V. (2017). Development of ontological support of constructive-synthesizing modeling of information systems. Eastern-European Journal of Enterprise Technologies, 6 (4 (90)), 58–69. doi: https://doi.org/10.15587/1729-4061.2017.119497

47. Skalozub, V., Ilman, V., Shynkarenko, V. (2018). Ontological support formation for constructive-synthesizing modeling of information systems development processes. Eastern-European Journal of Enterprise Technologies, 5 (4 (95)), 55–63. doi: https://doi.org/10.15587/1729-4061.2018.143968

48. Van der Aalst, W. (2016). Process mining: Data science in action. Springer. doi: https://doi.org/10.1007/978-3-662-49851-4

49. Leemans, S. J. J., Fahland, D., van der Aalst, W. M. P. (2018). Scalable process discovery and conformance checking. Software & Systems Modeling, 17 (2), 599–631. doi: https://doi.org/10.1007/s10270-016-0545-x