УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
сентябрь–октябрь 2020 Том 20 № 5 ISSN 2226-1494 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
September–October 2020 Vol. 20 No 5 ISSN 2226-1494 http://ntv.ifmo.ru/en/

УДК 004.424.4 doi: 10.17586/2226-1494-2020-20-5-714-721

МЕТОД ПОИСКА КЛОНОВ В ПРОГРАММНОМ КОДЕ

А.О. Осадчая, И.В. Исаев

Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
Адрес для переписки: osadchaya.alisa@gmail.com

Информация о статье
Поступила в редакцию 02.07.20, принята к печати 04.08.20
Язык статьи — русский

Ссылка для цитирования: Осадчая А.О., Исаев И.В. Метод поиска клонов в программном коде // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 5. С. 714–721. doi: 10.17586/2226-1494-2020-20-5-714-721

Аннотация

Предмет исследования. Исследованы существующие подходы и методы поиска клонов в программном коде. В результате исследования разработан метод, реализующий семантический подход поиска дублированных фрагментов, направленный на поиск клонов всех типов. **Метод.** Разработанный метод основан на анализе построенного по файлам исходного кода графа зависимостей программы. Для обнаружения дублированных фрагментов для каждого файла исходного кода генерируются графы зависимостей программы, узлы которых хешируются на основе свойств их содержимого. Пара узлов выбирается из каждого класса эквивалентности. Два изоморфных подграфа, которые включают в себя пару узлов, идентифицируются. Если пара клонов включена в другую пару, она удаляется из набора обнаруженных пар дублированных фрагментов. Набор клонов генерируется из пар дублированных фрагментов, совместно использующих те же изоморфные подграфы. Таким образом, происходит расширение пар клонов. **Основные результаты.** Для оценки эффективности работы разработанного метода поиска клонов проведено сравнение файлов для определения типов клонов, которые обнаруживает система, использующая данный метод, и тестирование работы на компонентах реальной системы. Выполнено сравнение результатов работы разработанной системы с реально существующими. **Практическая значимость.** Предложенный метод позволяет автоматизировать анализ исходных файлов. Поиск клонов в программном коде является приоритетным направлением в анализе кода. Обнаружение дублированных фрагментов позволяет бороться с недобросовестным копированием программного кода.

Ключевые слова
клоны в программном коде, дублирование кода, дублированные фрагменты, типы клонов кода, рефакторинг, анализ кода, повторное использование кода

doi: 10.17586/2226-1494-2020-20-5-714-721

SEARCH OF CLONES IN PROGRAM CODE

A.O. Osadchaya, I.V. Isaev

ITMO University, Saint Petersburg, 197101, Russian Federation
Corresponding author: osadchaya.alisa@gmail.com

Article info
Received 02.07.20, accepted 04.08.20
Article in Russian

For citation: Osadchaya A.O., Isaev I.V. Search of clones in program code. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2020, vol. 20, no. 5, pp. 714–721 (in Russian). doi: 10.17586/2226-1494-2020-20-5-714-721

Abstract

Subject of Research. The paper presents research of existing approaches and methods for the search of clones in the program code. As a result of the study, a method is developed that implements a semantic approach for the search of duplicated fragments focused on all kinds of clones. **Method.** The developed method is based on the analysis of the program dependency graph built from the source code files. To detect duplicate fragments, for each source code file dependency program graphs are generated with the nodes hashed on the basis of their content properties. Each pair of nodes is selected from each equivalence class, and two isomorphic subgraphs are identified that include a pair of nodes. If a pair of clones is included into another pair, it is removed from the set of the found pairs of duplicated fragments. A set of clones is generated from the pairs of duplicated fragments that share the same isomorphic subgraphs, that is, the pairs of clones are expanded. **Main Results.** To evaluate the efficiency of the developed method of searching for clones, the files have been compared for determination of the clone types that the system using this method detects, and

the testing has been performed on the real system components. The results of the developed system have been compared to the real ones. **Practical Relevance.** The proposed algorithm makes it possible to automate the analysis of source files. Detecting of clones in the program code is a priority direction in code analysis, since the detection of duplicate fragments provides for the fight against unscrupulous copying of program code.

Keywords
clones in program code, code duplication, duplicated fragments, code clone types, refactoring, code analysis, code reuse

Введение

В настоящее время сфера информационных технологий активно развивается, новые программные продукты появляются практически каждый день, а число строк программного кода ежегодно увеличивается в два раза [1]. Для упрощения процесса и времени разработки, а также поддержки программного обеспечения могут использоваться различные методы и инструменты для повышения эффективности. К таким подходам относятся: copy-paste programming, т. е. копирование и вставка фрагментов кода, которые незначительно изменяются под конкретные задачи, а также ментальные шаблоны, использование которых отрицательно влияет на наличие дублированных фрагментов в исходном коде.

В исходном коде программных продуктов содержится 10–15 % дублированных фрагментов [2]. Для успешного обнаружения клонов и эффективной борьбы с ними необходимо понимать, какие существуют типы клонов, и, в зависимости от того, с какими из них необходимо работать, уметь выявлять причины их возникновения в процессе разработки, а также избавляться от последствий их наличия различными методами. Выбор метода поиска клонов в программном коде базируется в основном на типах клонов, которые являются наиболее приоритетными для обнаружения.

Разработка инструментов для поиска дублированных фрагментов в настоящее время является перспективным направлением в сфере анализа программного кода. Такие инструменты позволяют проводить исследования с целью поиска недобросовестного копирования кода, в том числе для крупных корпораций.

Таким образом, целью работы является изучение существующих подходов и методов поиска клонов, а также разработка, на основе полученных результатов, собственного метода.

Существующие подходы

Клонами программного кода принято считать фрагменты кода, которые схожи по форматированию, синтаксису или семантике. Выделяют три основных типа клонов [3]:

1) T1 — фрагменты кода, которые могут отличаться только пробелами, комментариями и форматированием кода;
2) T2 — все клоны типа T1, а также фрагменты кода, которые могут также различаться: именами переменных; типами переменных; значениями переменных и констант;
3) T3 — все клоны типа T2, а также фрагменты кода, в которых могут быть добавлены или удалены инструкции и переменные.

Согласно данной классификации, клоны типа T1 являются наиболее простыми для обнаружения, клоны типа T3 — наиболее сложными, однако этот тип является приоритетным для обнаружения, поскольку учитывает семантику программы.

Существует множество причин появления в программном коде дублированных фрагментов, среди них можно выделить три основных [4].

Самой распространенной причиной наличия клонов в программном коде является недобросовестное повторное использование кода. Зачастую такой подход при написании программных продуктов обусловлен необходимостью увеличить скорость разработки. Например, при необходимости реализовать схожую функциональность в различных модулях программы методы копируются, вместо того чтобы выделить абстрактный класс и использовать унаследованные от него методы, меняя функциональность согласно поставленной задаче.

Также причиной появления в программном коде дублированных фрагментов является наличие у разработчиков-программистов паттернов разработки, в том числе ментальных шаблонов, приобретенных в процессе обучения и получения опыта при реализации проектов. К таким шаблонам относятся, например, обработка списков (фильтрация и свертка). Поскольку явного копирования при использовании данного подхода не осуществляется, дублированные фрагменты, полученные таким образом, являются более сложными для обнаружения, чем при прямом копировании, однако они считаются семантическими клонами.

При необходимости обеспечивать определенные требования к производительности реализуемого программного продукта также происходит копирование фрагментов кода. К оптимизациям для увеличения производительности относятся [5]: конвейеризация, разделение, разбиение и раскрутка цикла.

Первоначальная разработка программного продукта занимает меньшую часть времени разработки, около 20 %, сопровождение и поддержка — большую, около 80 % [6]. Таким образом, увеличение скорости разработки за счет дублирования фрагментов кода приводит к значительным проблемам и увеличению затрат на последующую поддержку разработанной системы, уменьшая тем самым ее качество.

Можно выделить несколько основных последствий наличия клонов в программном коде [7]: трудности при модификации кода и рефакторинге; появление новых ошибок при копировании фрагментов кода; увеличение размера исходного кода программного продукта.

Трудности при модификации и рефакторинге возникают по причине того, что при явном копировании фрагмента вместе с явными зависимостями между компонентами программы появляются неявные зави-

симости между дублированными фрагментами кода. Это означает, что при изменении одного фрагмента, необходимо обнаружить все, которые были скопированы из него, и изменить их.

Для минимизации дублирования фрагментов кода в процессе разработки программного продукта рекомендуется использовать различные методологии и правила, к примеру, DRY (Don't Repeat Yourself) [8] или «Правило трех» (Rule of Three) [9]. Такие правила подразумевают выполнение рефакторинга программного кода непосредственно после его написания.

Недобросовестное повторное использование кода приводит также к появлению новых ошибок в программном коде. Так, например, при копировании разработчик может забыть переименовать переменную или изменить инструкцию согласно поставленной задаче.

Наиболее очевидным последствием наличия дублированных фрагментов является увеличение размера программного кода. Согласно исследованиям, кратковременная человеческая память способна обеспечивать оптимальную работу с 7 ± 2 объектами [10], поэтому понимание и обработка кода большого размера занимает больше времени. Также к негативным последствиям увеличения размеров программного кода можно отнести увеличение времени компиляции, что является критическим фактором в высоконагруженных проектах.

При анализе программного кода с целью обнаружения дублированных фрагментов следует учитывать типы клонов, которые являются приоритетными для обнаружения, а также задачи, реализованные системой, для которой выполняется анализ. Существует пять основных подходов к обнаружению клонов в программном коде [11]:

1) текстовый — подход, основанный на обработке файлов исходного кода в виде текста, сравнение выполняется для строк или подстрок, для нормализации возможно применение языка TXL — особого универсального языка программирования [12];
2) лексический — подход, при котором сравнение выполняется для последовательностей токенов, полученных в результате обработки файлов исходного кода;
3) синтаксический — подход, основанный на сравнении абстрактных синтаксических деревьев (АСД) или деревьев разбора (ДР), которые строятся из файлов исходного программного кода;
4) семантический — подход, в котором в качестве объекта сравнения выступает граф зависимостей программы (ГЗП), который включает информацию о семантике программы, построенный в результате обработки файлов исходного кода;
5) метрический — подход к поиску клонов в программном коде, основанный на сравнении метрик, которые применяются к АСД или ГЗП исходного программного кода.

Результаты сравнения основных инструментов поиска клонов в программном коде представлены в таблице.

Анализ существующих инструментов поиска клонов выполнен на основе реализуемого алгоритма, представленного ниже в статье, и типов клонов, которые он позволяет обнаружить. В результате сравнения показаны основные преимущества и недостатки каждого из инструментов, реализующих вышеперечисленные подходы (таблица, поле «Подход»). По итогам анализа определено, что текстовый и лексический подходы не подходят для обнаружения клонов типа Т3, синтаксический — обладает низкой точностью при поиске клонов данного типа, метрический подход имеет низкую точность при поиске клонов всех типов, семантический — имеет достаточно высокую точность при поиске клонов всех типов. Поскольку наиболее сложными для обнаружения являются клоны типа Т3, он признан приоритетным для поиска. Таким образом, на основании вышеизложенного сделан научный вывод, что самым оптимальным является семантический подход.

Многие из рассмотренных инструментов не имеют в открытом доступе исходного кода или устарели, т. е. их невозможно установить и запустить, поэтому для дальнейшего сравнения с разработанным методом выбрано по одному инструменту каждого из подходов: текстового, лексического и синтаксического (DuDe, CCFinder и Deckard соответственно). Метрический подход исключен из дальнейшего анализа, так как обладает низкой точностью при поиске клонов всех типов.

Сущность предлагаемого метода

Реализованный инструмент поиска клонов в программном коде основан на сравнении ГЗП, построенных в результате обработки файлов исходного кода. Основное преимущество такого вида сравнения заключается в том, что для обнаружения становятся доступны несмежные клоны, т. е. те, которые расположены непоследовательно в программном коде, за счет возможности обнаружения Т3.

Для апробации метода использованы файлы исходного кода на языке Java, поскольку он является строго типизированным, что облегчает семантический анализ исходных файлов.

Предлагаемый метод включает в себя: синтаксический анализ; построение графа зависимостей программы; поиск клонов.

При разработке графа использована конструкция DirectedGraph библиотеки JGraphT, с ребрами, которые заданы переопределенным классом Edge, расширяющим возможности класса DefaultEdge и указывающим тип зависимости (относится к control flow или data flow).

Разработанная система PDGCloneDetector направлена на поиск клонов всех типов. При запуске программы аргументами командной строки задаются такие параметры, как путь к файлам для анализа и вывода результатов, минимальный размер клона и количество потоков, на которое будет разделено вычисление.

Предлагается следующий алгоритм работы.

1. Узлы ГЗП хешируются на основе свойств содержимого узлов.
2. Каждая пара узлов выбирается из каждого класса эквивалентности, и двух изоморфных подграфов, которые включают в себя пару узлов и идентифицируются.

Таблица. Сравнение инструментов поиска клонов

Подход	Метод	Инструмент	Алгоритм/ объект сравнения	Типы клонов	Преимущества	Недостатки
Текстовый	Построчное сравнение	Duploc	Последовательность совпадающих строк	T1	Скорость работы	Не подходит для поиска клонов типа T2, T3
		DuDe	Последовательность совпадающих строк и расширенные клоны	T1, T2, T3 (невысокая точность)	Скорость работы	Не подходит для поиска клонов типа T3
	Подстрочное сравнение	Johnson, sif	Отпечатки кода	T1, T2	Скорость работы	Не подходит для поиска клонов типа T3
	Использование языка TXL	NiCad	Разделение кода на фрагменты/Нормализованный код (TXL) (максимальное общее множество считается клоном)	T1, T2	Масштабируем	Не подходит для поиска клонов типа T3
Лексический	—	Dup, CCFinder	Совпадающие последовательности токенов (суффиксное дерево)	T1, T2	Позволяет находить клоны типов T1, T2 с высокой точностью	Не подходит для поиска клонов типа T3
		CP-Miner	Использует data mining для определения клонов	T1, T2	Работает медленнее других методов. Позволяет находить клоны типов T1, T2 с высокой точностью	Позволяет находить фрагменты кода с ошибками
		SABSM, RTF	Суффиксное дерево	T1, T2	Позволяет находить клоны типов T1, T2 с высокой точностью	Не подходит для поиска клонов типа T3
Синтаксический	—	Yang	АСД	T1, T2, T3	Показывает инструкции помимо отличающихся фрагментов кода	Низкая точность при поиске клонов типа T3, поскольку при изменении структуры кода меняется АСД
		Falke et al	Изоморфные поддеревья АСД и суффиксное дерево	T1, T2, T3	Высокая точность при поиске клонов типа T1, T2	Низкая точность при поиске клонов типа T3. Возможны пропуски клонов типа T2 при наличии ошибок переименовывания переменных
		DECKARD	ДР — кластеризованные на основе евклидового расстояния векторы поддеревьев	T1, T2, T3	За счет использования дерева разбора точность поиска клонов типа T3 выше, чем у других методов синтаксического подхода	Низкая точность для поиска клонов типа T3
		Clone Digger	Антиунификация/Классифицированные АСД, в которых поддеревья одинаковых классов заменяются вершинами класса	T1, T2, T3	Более высокая в сравнении с другими методами синтаксического подхода точность нахождения клонов	За счет антиунификации количество найденных клонов снижается

Таблица. (*продолжение*)

Подход	Метод	Инструмент	Алгоритм/ объект сравнения	Типы клонов	Преимущества	Недостатки
Семантический	—	Horwitz	ГЗП	T1, T2, T3	Высокая точность. Используется для поиска клонов в различных версиях программ	Высокая вычислительная сложность за счет количества вершин ГЗП
		GPLAG	Фильтрация / ГЗП	T1, T2, T3	Высокая точность. Работает быстрее других методов семантического подхода за счет применения фильтров	Высокая вычислительная сложность за счет количества вершин ГЗП. Есть ограничения по размеру исходного кода
		Gabel et al	Инструменты DECKARD / ГЗП	T1, T2, T3	Высокая точность, масштабируемость	Высокая вычислительная сложность за счет количества вершин ГЗП. Возможны потери при масштабируемости: T3, ошибки при поиске
		CBCD	ГЗП и база фрагментов кода, содержащего ошибки	T1, T2, T3	Высокая точность. Позволяет находить клоны кода, в которых содержатся ошибки	Высокая вычислительная сложность за счет количества вершин ГЗП
Метрический	—	Mayrand et al, Patenaude et al, Kontogiannis et al, Kodhai et al, Li et al	Метрики на основе АСД	T1, T2, T3	Высокая скорость работы, масштабируемость	Низкая точность

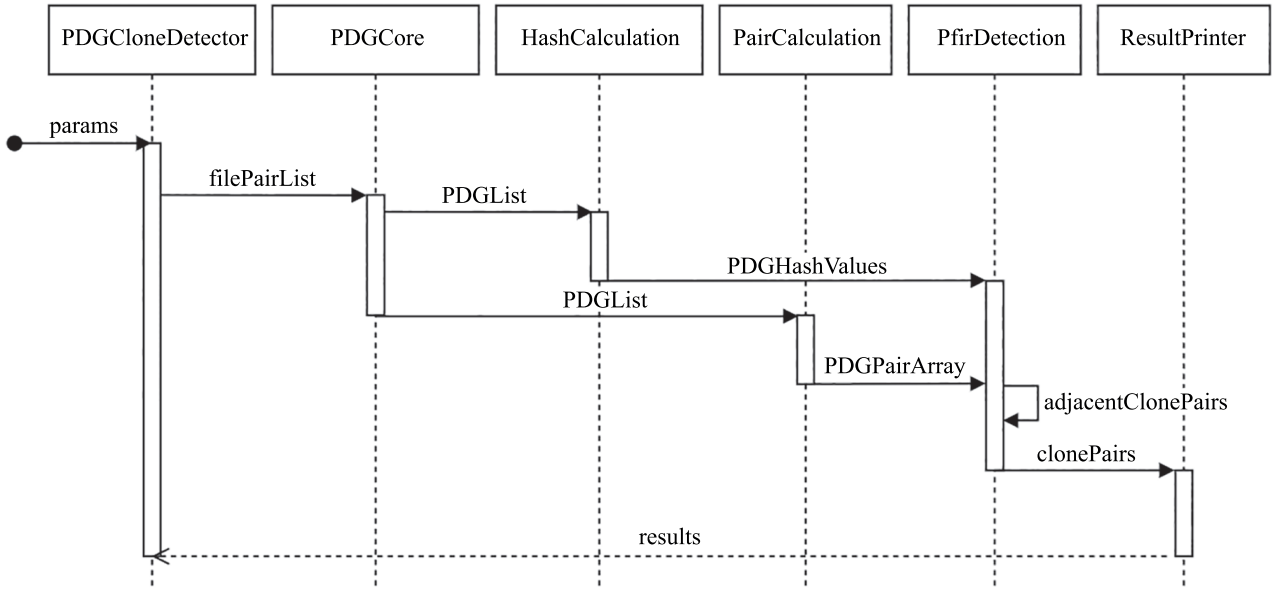

Рис. 1. Диаграмма последовательности PDGCloneDetector (params — аргументы командной строки, results — результаты работы алгоритма: время работы, количество найденных клонов)

3. Если пара клонов включена в другую пару клонов, она удаляется из набора обнаруженных пар клонов.
4. Набор клонов генерируется из пар клонов, совместно использующих те же изоморфные подграфы, т. е. происходит расширение пар клонов.

Диаграмма последовательности разработанного подхода представлена на рис. 1.

При запуске программы в PDGCloneDetector передаются аргументы командной строки, из указанной директории загружаются файлы, которые формируют список пар файлов (filePairList) в PDGCore, для которых строятся ГЗП (PDGList). Для каждого графа зависимостей программы выполняется хеширование узлов (PDGHashValues) в HashCalculation для последующего сравнения (PairCalculation) и определения идентичных пар узлов (PairDetection). После нахождения каждой пары дублированных фрагментов (PDGPairArray) выполняется проверка на возможность расширения клона (adjacentClonePairs). После обработки всех файлов исходного кода результаты (clonePairs) помещаются в ResultPrinter.

Апробация предлагаемого метода

Для оценки эффективности работы разработанного инструмента поиска клонов проведено несколько видов тестирования: сравнение файлов для определения типов клонов, которые обнаруживает система, и тестирование работы на компонентах реальной системы (пакет сервисов проекта Энерготрейдинг компании ООО «НеМо», содержащий 378 файлов). Выполнено сравнение результатов работы разработанной системы с реально существующими (DuDe [13], CCFinder [14], Deckard [15]).

По результатам тестирования системы обнаружены клоны всех трех типов, при тестировании на системе Энерготрейдинг обнаружено 8 672 клона.

Сравнение результатов работы PDGCloneDetector с DuDe, CCFinder и Deckard представлено на рис. 2.

Анализ инструментальных средств основан на определении количества найденных клонов и времени работы системы. Время работы PDGCloneDetector достаточно велико — 1602 с, что намного превосходит наилучший по времени работы инструмент DuDe (время работы 1 с). Однако, учитывая типы найденных клонов и сложность обнаружения клонов типа Т3, наиболее приоритетным фактором оптимальности работы системы является количество найденных клонов. PDGCloneDetector показал наилучший результат среди исследуемых инструментов — 8 672 клона, в то время как остальные инструменты, в основном ориентированные на поиск клонов типа Т1 и Т2, DuDe, CCFinder и Deckard, обнаружили 5 966, 5 921 и 1 178 клонов соответственно. Исходя из вышеперечисленного, можно сделать вывод, что по приоритетному фактору сравнения разработанный метод имеет наилучший результат, но производительность требует доработки, поскольку алгоритм обработки большого количества файлов исходного кода, построения ГЗП и сравнения узлов занимает большое количество времени.

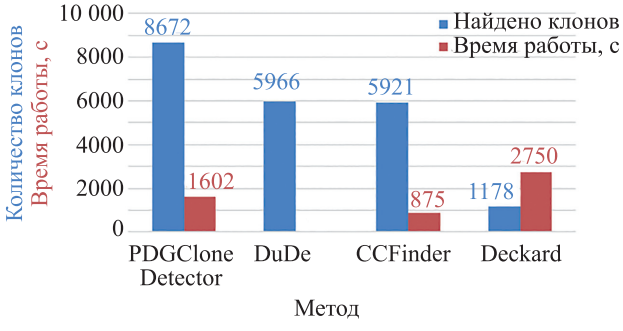

Рис. 2. Сравнительная диаграмма работы PDGCloneDetector, DuDe, CCFinder и Deckard

Заключение

В работе проведено исследование клонов программного кода, определение основных типов клонов, причин и последствий их наличия в программном коде. В ходе анализа существующих подходов к обнаружению дублированных фрагментов в коде выделен наиболее приоритетный, по которому была разработана система. Выполнено тестирование результатов работы разработанного инструмента и сравнение с реально существующими системами.

Алгоритм построения графа зависимостей программы и сравнения полученных узлов занимает большую часть времени работы программы ($O(n^3)$, где O — время работы алгоритма; n — количество узлов графа зависимостей программы), что требует доработки с целью увеличения производительности системы. Перспективы доработки заключаются в интегрировании разработанного инструмента в реально существующие системы, а также в реализации плагина для интегрированных сред для удобства использования разработчиками.

Литература

1. Deshpande A., Riehle D. The total growth of open source // IFIP International Federation for Information Processing. 2008. V. 275. P. 197–209. doi: 10.1007/978-0-387-09684-1_16
2. Kapser C., Godfrey M.W. Toward a taxonomy of clones in source code: A case study // Proc. of the Workshop Evolution of Large-scale Industrial Software Applications (ELISA). 2003. P. 67–78.
3. Саргсян С.С. Методы поиска клонов кода и семантических ошибок на основе семантического анализа программы: диссертация на соискание ученой степени кандидата физико-математических наук. Москва: Институт системного программирования Российской академии наук, 2016. С. 10–22.
4. Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010. 552 с.
5. Bacon D.F., Graham S.L., Sharp O.J. Compiler transformations for high-performance computing // ACM Computing Surveys. 1994. V. 26. N 4. P. 345–420. doi: 10.1145/197405.197406
6. Glass R.L. Frequently forgotten fundamental facts about software engineering // IEEE Software. 2001. V. 18. N 3. P. 110–112. doi: 10.1109/MS.2001.922739
7. Ахин М.Х., Ицыксон В.М. Обнаружение клонов исходного кода: теория и практика // Системное программирование. 2010. Т. 5. № 1. С. 145–163.
8. Hunt A., Thomas D. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley Professional, 1999. 352 p.
9. Fowler M., Beck K., Brant J., Opdyke W., Roberts D. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999. 464 p.
10. Miller G.A. The magical number seven, plus or minus two: some limits on our capacity for processing information // Psycological Review. 1956. V. 63. N 2. P. 81–97. doi: 10.1037/h0043158
11. Ducasse S., Rieger M., Demeyer S. A language independent approach for detecting duplicated code // Proc. 15th International Conference on Software Maintenance (ICSM). 1999. P. 109–118. doi: 10.1109/ICSM.1999.792593
12. Cordy J.R. The TXL source transformation language // Science of Computer Programming. 2006. V. 61. N 3. P. 190–210. doi: 10.1016/j.scico.2006.04.002
13. Wettel R., Marinescu R. Archeology of code duplication: Recovering duplication chains from small duplication fragments // Proc. 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005). 2005. P. 6370. doi: 10.1109/SYNASC.2005.20
14. Livieri S., Higo Y., Matsushita M., Inoue K. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder // Proc. 29th International Conference on Software Engineering (ICSE). 2007. P. 106–115. doi: 10.1109/ICSE.2007.97
15. Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones // Proc. 29th International Conference on Software Engineering (ICSE). 2007. P. 96–105. doi: 10.1109/ICSE.2007.30

References

1. Deshpande A., Riehle D. The total growth of open source. *IFIP International Federation for Information Processing*, 2008, vol. 275, pp. 197–209. doi: 10.1007/978-0-387-09684-1_16
2. Kapser C., Godfrey M.W. Toward a taxonomy of clones in source code: A case study. *Proc. of the Workshop Evolution of Large-scale Industrial Software Applications (ELISA)*, 2003, pp. 67–78.
3. Sargsian S.S. *Search methods for code clones and semantic errors based on semantic program analysis*. Dissertation for the degree of candidate of physical and mathematical sciences. Moscow, ISPRAS, 2016, p. 10–22. (in Russian)
4. Karpov Iu.G. *Model Checking. Verification of Parallel and Distributed Software Systems*. St. Petersburg, BHV Publ., 2010, 552 p. (in Russian)
5. Bacon D.F., Graham S.L., Sharp O.J. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 1994, vol. 26, no. 4, pp. 345–420. doi: 10.1145/197405.197406
6. Glass R.L. Frequently forgotten fundamental facts about software engineering. *IEEE Software*, 2001, vol. 18, no. 3, pp. 110–112. doi: 10.1109/MS.2001.922739
7. Akhin M.Kh., Itcykson V.M. Source code clone detection: theory and practice. *Sistemnoe Programmirovanie*, 2010, vol. 5, no. 1, pp. 145–163. (in Russian)
8. Hunt A., Thomas D. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999, 352 p.
9. Fowler M., Beck K., Brant J., Opdyke W., Roberts D. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999, 464 p.
10. Miller G.A. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psycological Review*, 1956, vol. 63, no. 2, pp. 81–97. doi: 10.1037/h0043158
11. Ducasse S., Rieger M., Demeyer S. A language independent approach for detecting duplicated code. *Proc. 15th International Conference on Software Maintenance (ICSM)*, 1999, pp. 109–118. doi: 10.1109/ICSM.1999.792593
12. Cordy J.R. The TXL source transformation language. *Science of Computer Programming*, 2006, vol. 61, no. 3, pp. 190–210. doi: 10.1016/j.scico.2006.04.002
13. Wettel R., Marinescu R. Archeology of code duplication: Recovering duplication chains from small duplication fragments. *Proc. 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005)*, 2005, pp. 6370. doi: 10.1109/SYNASC.2005.20
14. Livieri S., Higo Y., Matsushita M., Inoue K. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder. *Proc. 29th International Conference on Software Engineering (ICSE)*, 2007, pp. 106–115. doi: 10.1109/ICSE.2007.97
15. Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones. *Proc. 29th International Conference on Software Engineering (ICSE)*, 2007, pp. 96–105. doi: 10.1109/ICSE.2007.30

Авторы

Осадчая Алиса Олеговна — студент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ORCID ID: 0000-0002-7888-0998, osadchaya.alisa@gmail.com
Исаев Илья Владимирович — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, Scopus ID: 57191272743, ORCID ID: 0000-0003-4736-0129, ivisaev@itmo.ru

Authors

Alisa O. Osadchaya — Student, ITMO University, Saint Petersburg, 197101, Russian Federation, ORCID ID: 0000-0002-7888-0998, osadchaya.alisa@gmail.com
Ilia V. Isaev — Postgraduate, ITMO University, Saint Petersburg, 197101, Russian Federation, Scopus ID: 57191272743, ORCID ID: 0000-0003-4736-0129, ivisaev@itmo.ru