

Ф. Г. Жучков¹, соискатель, philipgermanovich@yandex.ru,
Е. Р. Брюханова^{2, 3}, канд. техн. наук, доц., ст. системный аналитик, evgbryuhanova@gmail.com, **О. В. Колосова**¹, д-р техн. наук, проф. Высшей школы технологического предпринимательства, okolosova@spbstu.ru,
А. М. Хахина¹, д-р техн. наук, проф. высшей школы программной инженерии, hahina_am@spbstu.ru,
О. А. Антамошкин³, д-р техн. наук, зав. кафедрой программной инженерии, oleslav@mail.ru
¹ Санкт-Петербургский политехнический университет Петра Великого,
² АО «РУТ КОД», Москва,
³ Сибирский федеральный университет, Красноярск

Управляемое улучшение программных агентов на основе больших языковых моделей с использованием функции пригодности

*Поступила в редакцию 09.04.2026
Принята к публикации 16.06.2026*

Рассмотрена задача разработки программных агентов на основе больших языковых моделей, способных автоматически выполнять отдельные операции по изменению программного кода, проверке результата и накоплению опыта предыдущих действий.

Предложен подход к управляемому улучшению программных систем, при котором агент формирует гипотезы изменений, вносит модификации в кодовую базу, запускает тестовый контур и оценивает полученный результат с использованием функции оценки качества. Управляемость процесса обеспечивается за счет фиксированных условий эксперимента, автоматической проверки изменений и сопоставления результатов различных стратегий поведения агента.

Для экспериментальной оценки разработан стенд RSI-Bench, предназначенный для сравнения стратегий поведения программных агентов при решении задач модификации кода. В качестве основной метрики использовалась успешность прохождения автоматических тестов. Дополнительно фиксировались вычислительные затраты и число итераций, необходимых для получения результата.

Проведенные эксперименты показали, что в рассмотренных условиях качество решений зависит не только от выбранной стратегии поведения агента, но и от базовой языковой модели, конкретной реализации агента и случайной вариативности между запусками. Установлено, что стратегии самомодификации на уровне текстовых инструкций дают ограниченный и нестабильный эффект. Полученные результаты могут использоваться при разработке систем автоматизированной поддержки программной инженерии и средств оценки качества изменений программного кода.

Ключевые слова: *программная инженерия, большие языковые модели, программные агенты, генерация программного кода, самомодификация, функция пригодности, поисковая оптимизация, оценка качества программных систем*

Для цитирования: Жучков Ф. Г., Брюханова Е. Р., Колосова О. В., Хахина А. М., Антамошкин О. А. Управляемое улучшение программных агентов на основе больших языковых моделей с использованием функции пригодности // Программная инженерия. 2026. Том 17, № 10. С. 535—541. DOI: 10.17587/prin.17.535-541.

Введение

В последние годы большие языковые модели (*Large Language Model*, LLM) активно применяются для автоматизации задач программной инженерии. На их основе создаются программные агенты, способные анализировать кодовую базу, генерировать изменения и проверять результаты с использованием тестового контура.

Такие агенты [1] используются для решения практических задач, включая исправление ошибок, добавление функциональных возможностей и автоматизацию отдельных этапов разработки. Однако существующие решения, как правило, ориентированы на выполнение отдельных задач и не обеспечивают устойчивого процесса последовательного улучшения программной системы.

Указанные решения можно условно разделить на несколько направлений. Во-первых, это агенты [2], ориентированные на решение конкретных задач, например, исправление дефектов в программном коде. Они используют анализ кодовой базы и тестирование для проверки корректности предлагаемых изменений. Во-вторых, развиваются методы [3], основанные на использовании текстовой обратной связи и накоплении опыта, позволяющие частично улучшать качество решений за счет повторного использования предыдущих результатов.

Отдельное направление исследований связано с использованием структурированных стратегий рассуждения [4] и поиска решений [5], позволяющих агентам рассматривать несколько альтернативных вариантов выполнения задачи. Такие подходы направлены на повышение качества принимаемых решений и снижение вероятности ошибочных изменений, однако их эффективность существенно зависит от особенностей реализации агента и организации взаимодействия с языковой моделью.

Методологическую основу для формализации процесса улучшения программных систем составляет направление *Search-Based Software Engineering* (SBSE) [6–8], в рамках которого задачи программной инженерии рассматриваются как задачи оптимизации в пространстве решений. В этих подходах используется функция оценки качества, позволяющая сравнивать различные варианты решений и выбирать наиболее эффективные из них.

В отечественных исследованиях также рассматривались вопросы проектирования программных систем, мультиагентных архитектур [9] и методов управления распределенными вычислительными средами [10]. Это позволяет рассматривать исследуемую задачу не только как развитие LLM-

агентов, но и как продолжение работ по автоматизации и оценке качества программных систем.

Несмотря на активное развитие подобных подходов, эффективность стратегий самомодификации и модификации управляющих инструкций [11] остается ограниченной и существенно зависит от конкретной реализации агента. Дополнительной сложностью является высокая вариативность результатов: качество работы агентов может существенно зависеть от используемой языковой модели, конкретной реализации алгоритма, выбранной стратегии и случайных факторов. При этом в большинстве работ анализируется влияние только одного из этих факторов, что затрудняет получение обобщенных выводов. Кроме того, высокая вариативность поведения агентов поднимает вопрос не только об эффективности отдельных стратегий, но и об устойчивости и управляемости поведения таких систем. Это определяет постановку задачи настоящего исследования.

Постановка задачи исследования

В настоящей работе рассмотрена задача разработки программного агента на основе большой языковой модели, способного автоматически вносить изменения в программную систему, проверять их корректность с использованием тестового контура и оценивать качество полученных результатов.

Пусть задана программная система, представленная в виде исходного кода и набора автоматических тестов. Требуется построить агента, который в процессе работы последовательно формирует гипотезы изменений, модифицирует кодовую базу и проверяет результат. Изменение считается успешным, если оно приводит к корректному выполнению тестов и достижению требуемой функциональности.

Процесс работы агента рассматривается как последовательность итераций, на каждой из которых выполняются следующие действия: анализ текущего состояния системы, генерация гипотезы изменения, внесение модификации и проверка результата. Для отбора изменений используется функция оценки качества, позволяющая сравнивать различные варианты решений.

С учетом высокой вариативности поведения LLM-агентов возникает необходимость исследования факторов, влияющих на качество их работы. В частности, требуется оценить влияние следующих факторов: используемой языковой модели, программной реализации агента, стратегии взаимодействия с моделью, случайной вариативности между запусками.

Таким образом, задача работы заключается в разработке подхода к улучшению программных систем с использованием LLM-агентов и проведении экспериментального исследования, направленного на выявление факторов, определяющих качество их работы.

Методы исследования

В работе процесс улучшения программной системы рассматривается как последовательность изменений программного состояния, выполняемых агентом на основе большой языковой модели.

Пусть программная система находится в состоянии S_t , включающем исходный код, тестовый контур и сопутствующие артефакты разработки. На каждой итерации агент формирует гипотезу изменения H_t , на ее основе создает изменение ΔS_t , вносит его в кодовую базу и выполняет проверку результата. Переход системы в новое состояние описывается следующим образом: $S_{t+1} = S_t + \Delta S_t$, где ΔS_t — фактически внесенное изменение программной системы.

После внесения изменений выполняются тестирование и оценка результата. В общем случае качество состояния программной системы может определяться с использованием функции пригодности: $F(S) = \sum_{i=1}^n w_i f_i(S)$, где $f_i(S)$ — частные показатели качества; w_i — соответствующие весовые коэффициенты; n — число учитываемых показателей качества. Значение n определяется составом показателей, включенных в конкретную процедуру оценки.

В рамках данной работы функция пригодности рассматривается как обобщенная модель оценки качества изменений. При проведении экспериментального исследования в стенде RSI-Bench использовалась упрощенная бинарная метрика успешности выполнения задачи, определяемая по результату прохождения автоматических тестов.

Архитектура агента включает анализ текущего состояния программной системы, генерацию гипотез изменения, модификацию программного кода, запуск тестового контура, оценку результата и сохранение информации о предыдущих действиях.

Работа агента организована в виде итерационного цикла, в котором результат предыдущего шага используется при формировании следующего изменения.

Для оценки влияния различных факторов на качество решений в эксперименте рассматривались: используемая языковая модель, программная реализация агента, стратегия взаимодействия с моделью, случайная вариативность между запусками. Это позволило исследовать вклад отдельных факторов в итоговое качество решений и сопоставить эффективность различных стратегий поведения агентов.

Экспериментальный стенд RSI-Bench

Для экспериментальной оценки был разработан стенд RSI-Bench, предназначенный для сравнительного исследования поведения LLM-агентов при решении задач модификации программного кода.

В качестве объекта модификации использовалась Python-библиотека PrettyTable — компактный проект с четко определенным программным интерфейсом, развитым тестовым покрытием и отсутствием внешних зависимостей. Выбор обусловлен тем, что задачи модификации данной библиотеки не требуют узкоспециальных знаний предметной области и позволяют отделить влияние стратегии поведения агента от влияния сложности прикладной задачи. Набор из десяти задач средней сложности обеспечивал достаточную вариативность результатов для сравнительного анализа.

Один цикл работы стенда включал постановку задачи, создание изолированной рабочей копии репозитория для каждой задачи, выполнение агентом серии действий по чтению и изменению кода, запуск автоматической проверки с использованием pytest и фиксацию результата. Основным показателем являлось число задач из десяти, для которых после работы агента была успешно пройдена автоматическая проверка. Дополнительно фиксировались стоимость выполнения и число раундов взаимодействия с языковой моделью.

Архитектурно-технологический каркас стенда и исследуемые стратегии поведения агента приведены ниже.

Набор задач	Десять задач модификации Python-библиотеки PrettyTable
Среда исполнения	Изолированная рабочая копия репозитория для каждой задачи
LLM-интерфейс	Генерация действий на основе модели Claude Sonnet 4.5
Тестовый контур (pytest)	Внешняя автоматическая проверка корректности решения
Логирование	Фиксация результата (pass/fail), стоимости и числа раундов
Базовый агент	Без механизмов самомодификации; результат определяется моделью и исходной инструкцией
Текстовая память (Scratchpad)	Агент сохраняет текстовые заметки между задачами и использует их в дальнейшем
Инъекция кода (Code diffs)	В инструкцию агента встраиваются фрагменты успешных изменений из предыдущих задач
Самоперезапись инструкций (Prompt surgery)	После каждой задачи агент формирует обновленную версию собственной инструкции
Смена стратегии (Checkpoint agent)	На середине выполнения агент оценивает прогресс и может перезапустить решение

Такое устройство стенда позволяет сопоставлять стратегии поведения агента в контролируемой постановке: входные задачи, правила запуска, тестовый контур и критерий успешности задаются единым образом, а основным изменяемым параметром выступает организация поведения агента.

Экспериментальная оценка

Эксперименты проводились на описанном выше стенде RSI-Bench.

В ходе исследования сравнивались варианты организации поведения агента, представленные выше. Каждая конфигурация запускалась до 3 раз при одинаковых входных условиях, ограничениях по числу итераций и фиксированном вычислительном бюджете для оценки устойчивости результатов.

В качестве основной метрики использовалась бинарная оценка успешности выполнения задачи (pass/fail), определяемая по результату прохождения автоматических тестов. Дополнительно фиксировались вычислительные затраты, число итераций, необходимых для получения результата, а также устойчивость поведения между различными запусками.

Результаты эксперимента представлены в табл. 1. Проведенное исследование показало, что использование текстовой памяти обеспечивает лишь ограниченное улучшение результатов. Применение примеров модификации кода в ряде случаев приводило к ухудшению качества решений, что может быть связано с закреплением неудачных шаблонов изменения программного кода и снижением вариативности поиска решений.

Особый интерес представляет стратегия *Checkpoint agent*. Две независимые реализации одной и той же архитектурной идеи продемонстрировали существен-

но различающиеся результаты (5,0 и 9,67 успешных задач из 10 соответственно). Полученный эффект оказался сопоставимым или превышающим различия между исследуемыми стратегиями поведения агентов, что указывает на значительное влияние особенностей программной реализации.

Изменение управляющих инструкций не обеспечило устойчивого прироста качества при увеличении вычислительных затрат. В рамках проведенного эксперимента различия между реализациями агента оказались более выраженными, чем различия между исследуемыми стратегиями поведения.

Дополнительно было проведено сравнение поведения базового агента и агента со сменой стратегии на нескольких языковых моделях (табл. 2).

Было отмечено, что для модели, показавшей нулевой результат в данной серии экспериментов (DeepSeek V3), ни одна из стратегий не обеспечивала ненулевого результата, что может указывать на существование порога способностей, ниже которого стратегии самомодификации не оказывают заметного эффекта.

Анализ экспериментальных данных позволяет предположить, что вклад различных факторов в изменчивость результатов неодинаков. Наиболее заметные различия в проведенных экспериментах были связаны с выбором базовой языковой модели и особенностями программной реализации агента. Влияние стратегии взаимодействия с моделью оказалось менее устойчивым: отдельные стратегии демонстрировали сопоставимые или более низкие результаты по сравнению с базовым агентом. Случайная вариативность между запусками также присутствовала, однако в рамках проведенной серии экспериментов не являлась основным источником различий.

Полученные наблюдения следует рассматривать как качественную интерпретацию результатов, полученных на ограниченном наборе задач, моделей и реализаций. Для количественной оценки вклада

Таблица 1

Результаты эксперимента RSI-Bench

Стратегия	Средний результат, задач из десяти	Стоимость, долл. США/задача
Базовый агент	7,33	0,88
<i>Scratchpad</i>	6,0	≈1,00
<i>Code diffs</i>	5,0	0,93
<i>Prompt surgery</i>	7,33	1,09
<i>Checkpoint agent</i> (реализация A)	5,0	0,22
<i>Checkpoint agent</i> (реализация B)	9,67	0,27

Таблица 2

Сравнение результатов на различных языковых моделях

Модель	Базовый агент, задач из десяти	Checkpoint agent, задач из десяти	Стоимость, долл. США/задача
Claude Sonnet 4.5	7,67 (±1,53)	5,0 (±0,0)	0,18
GPT-4o	3,67 (±1,53)	2,33 (±1,15)	0,06
DeepSeek V3	0,0 (±0,0)	0,0 (±0,0)	0,008
Примечание: сравнение моделей выполнено в рамках отдельной серии экспериментов.			

отдельных факторов необходимы дополнительные эксперименты с расширенным набором конфигураций и последующей статистической обработкой результатов.

Можно также отметить, что различные задачи обладали неодинаковой способностью различать качество стратегий поведения агентов. Часть задач демонстрировала практически одинаковые результаты для всех вариантов агентов, тогда как на других задачах различия между стратегиями становились более заметными. Это может указывать на целесообразность более формализованного подхода к построению тестовых наборов и оценке поведения программных агентов в задачах программной инженерии.

Полученные результаты подтверждают ограниченность методов самомодификации, основанных исключительно на изменении текстовых инструкций, и показывают необходимость дальнейшего развития методов оценки и отбора изменений программного кода.

Анализ результатов

Полученные данные указывают на существенную роль программной реализации агента. Две реализации стратегии *Checkpoint agent* дали заметно различающиеся результаты, что показывает чувствительность таких систем к деталям реализации и организации взаимодействия с языковой моделью.

Дополнительно было отмечено, что тестовые задачи различаются по способности выявлять различия между стратегиями. Часть задач решалась большинством конфигураций, тогда как различия между агентами проявлялись преимущественно на более сложных заданиях.

Полученные результаты также позволяют сделать вывод о том, что изменение текстовых инструкций само по себе не обеспечивает устойчивого механизма улучшения качества решений. Более перспективным направлением представляется использование формализованных методов оценки изменений, а также применение механизмов адаптации, затрагивающих не только управляющие инструкции, но и параметры модели или структуры взаимодействия агента с программной средой.

Следует отметить, что проведенное исследование имеет ряд ограничений. Эксперименты выполнялись на ограниченном наборе задач и библиотек, а основная метрика оценки основывалась на бинарной проверке корректности выполнения тестов. Кроме того, часть результатов носит качественный характер и требует дальнейшей ста-

тистической проверки на более широком наборе конфигураций и сценариев использования программных агентов.

Заключение

Рассмотрена задача разработки программных агентов на основе больших языковых моделей, способных автоматически вносить изменения в программный код и выполнять проверку полученных результатов с использованием тестового контура.

Предложен подход к улучшению программных систем, основанный на последовательном формировании гипотез изменений, модификации кодовой базы и оценке результатов выполнения. Для экспериментального исследования разработан стенд RSI-Bench, позволяющий сравнивать различные варианты реализации агентов и стратегии их поведения в контролируемых условиях.

Проведенные эксперименты показали, что в рассматриваемой постановке качество решений в значительной степени зависит от выбора базовой языковой модели и особенностей реализации агента. При этом методы самомодификации, основанные на изменении текстовых инструкций, обеспечивают ограниченный и нестабильный эффект. Также отмечено, что различные задачи обладают неодинаковой способностью различать качество стратегий поведения агентов, что целесообразно учитывать при построении тестовых наборов и экспериментальных стендов.

Полученные результаты подтверждают необходимость дальнейшего развития методов формализованной оценки поведения программных агентов и разработки более устойчивых подходов к автоматическому улучшению программных систем.

Практическая значимость работы заключается в возможности использования предложенного подхода и экспериментального стенда при исследовании программных агентов, предназначенных для автоматизации задач программной инженерии. Кроме того, полученные результаты могут использоваться при разработке систем оценки качества изменений программного кода и построении инфраструктуры тестирования LLM-агентов.

Полученные результаты показывают, что при оценке программных агентов необходимо учитывать не только факт успешного выполнения задачи, но и устойчивость результата при изменении реализации агента, стратегии его поведения и используемой языковой модели.

Перспективы дальнейших исследований связаны с расширением набора экспериментальных

задач, проведением более масштабного статистического анализа факторов вариативности, а также с исследованием управляемости программных агентов в контролируемых процессах программной инженерии.

Список литературы

1. Yang J., Jimenez C. E., Wettig A. et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering // arXiv preprint. 2024. arXiv:2405.15793. DOI: 10.48550/arXiv.2405.15793.
2. Zhang Y., Ruan H., Fan Zh., Roychoudhury A. AutoCodeRover: Autonomous Program Improvement // Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'24). New York: ACM, 2024. P. 1592—1604. DOI: 10.1145/3650212.3680384.
3. Shinn N., Cassano F., Gopinath A., Narasimhan K., Yao Sh. Reflexion: Language Agents with Verbal Reinforcement Learning // Advances in Neural Information Processing Systems. 2023. Vol. 36. P. 8634—8652. DOI: 10.52202/075280-0377.
4. Besta M., Blach N., Kubicek A. et al. Graph of Thoughts: Solving Elaborate Problems with Large Language Models // Proceedings of the AAAI Conference on Artificial Intelligence. 2024. Vol. 38, no. 16. P. 17682—17690. DOI: 10.1609/aaai.v38i16.29720.
5. Zhou A., Yan K., Shlapentokh-Rothman M. et al. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models // Proceedings of Machine Learning Research. 2024. Vol. 235. P. 62138—62160.
6. Harman M. The Current State and Future of Search-Based Software Engineering // Future of Software Engineering (FOSE 2007). Washington: IEEE Computer Society, 2007. P. 342—357. DOI: 10.1109/FOSE.2007.29.
7. Harman M., Mansouri S. A., Zhang Y. Search-Based Software Engineering: Trends, Techniques and Applications // ACM Computing Surveys. 2012. Vol. 45, no. 1. Article 11. DOI: 10.1145/2379776.2379787.
8. McMinn P. Search-Based Software Test Data Generation: A Survey // Software Testing, Verification and Reliability. 2004. Vol. 14, no. 2. P. 105—156. DOI: 10.1002/stvr.294.
9. Антамошкин О. А. Программная инженерия. Теория и практика: учеб. Красноярск: Сибирский федеральный университет, 2012. 247 с.
10. Antamoshkin O. A., Antamoshkina O. A., Smirnov N. A. Multi-agent automation system for monitoring, forecasting and managing emergency situations // IOP Conference Series: Materials Science and Engineering. 2016. Vol. 122. Article 012003. DOI: 10.1088/1757-899X/122/1/012003.
11. Sartaj H., Ali Sh., Arcaini P., Arcuri A. Search-Based Software Engineering and AI Foundation Models: Current Landscape and Future Roadmap // arXiv preprint. 2025. arXiv: 2505.19625. DOI: 10.48550/arXiv.2505.19625.

Controlled Improvement of Software Agents Based on Large Language Models Using a Fitness Function

F. G. Zhuchkov¹, Doctoral Candidate, philipgermanovich@yandex.ru,

E. R. Bryukhanova^{2, 3}, Cand. Sc. (Eng.), Associate Professor, Senior Systems Analyst, evgbryuhanova@gmail.com,

O. V. Kolosova¹, D. Sc. (Eng.), Professor of the Higher School of Technological Entrepreneurship, okolosova@spbstu.ru,

A. M. Khakhina¹, D. Sc. (Eng.), Professor of the Higher School of Software Engineering, hahina_am@spbstu.ru,

O. A. Antamoshkin³, D. Sc. (Eng.), Head of the Department of Software Engineering, oleslav@mail.ru

¹ Peter the Great St. Petersburg Polytechnic University, Saint Petersburg, Russian Federation,

² RUT CODE JSC, Moscow, Russian Federation,

³ Siberian Federal University, Krasnoyarsk, Russian Federation

Corresponding author:

Oleslav A. Antamoshkin, D. Sc. (Eng.), Head of the Department of Software Engineering, Siberian Federal University, 660041, Krasnoyarsk, Russian Federation
E-mail: oleslav@mail.ru

Received on April 09, 2026

Accepted on June 16, 2026

The paper addresses the problem of developing software agents based on large language models capable of automatically performing individual operations related to software code modification, result verification, and accumulation of experience from previous actions.

An approach to the controlled improvement of software systems is proposed, in which an agent generates hypotheses of code modifications, applies changes to the codebase, executes a testing pipeline, and evaluates the obtained result using a quality assessment function. Process controllability is ensured through fixed experimental conditions, automated verification of modifications, and comparison of different agent behavior strategies.

For experimental evaluation, the RSI-Bench framework was developed to compare strategies of software agent behavior in code modification tasks. The primary evaluation metric was the success rate of automated test execution. Additionally, computational cost and the number of iterations required to obtain a result were recorded.

The conducted experiments demonstrated that solution quality depends not only on the selected agent behavior strategy, but also on the underlying language model, the specific agent implementation, and stochastic variability between runs. It was found that self-modification strategies implemented at the level of textual instructions provide a limited and unstable effect. The obtained results may be used in the development of automated software engineering support systems and approaches for assessing the quality of software code modifications.

Keywords: software engineering, large language models, software agents, software code generation, self-modification, fitness function, search-based optimization, software quality assessment

For citation: Zhuchkov F. G., Bryukhanova E. R., Kolosova O. V., Khakhina A. M., Antamoshkin O. A. Controlled Improvement of Software Agents Based on Large Language Models Using a Fitness Function, *Programmnaya Ingeneria*, 2026, vol. 17, no. 10, pp. 535—541. DOI: 10.17587/prin.17.535-541.

References

1. Yang J., Jimenez C. E., Wettig A. et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv preprint*, 2024, arXiv:2405.15793. DOI: 10.48550/arXiv.2405.15793.
2. Zhang Y., Ruan H., Fan Zh., Roychoudhury A. Auto-CodeRover: Autonomous Program Improvement, *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24)*, New York, ACM, 2024. P. 1592—1604. DOI: 10.1145/3650212.3680384.
3. Shinn N., Cassano F., Gopinath A., Narasimhan K., Yao Sh. Reflexion: Language Agents with Verbal Reinforcement Learning, *Advances in Neural Information Processing Systems*, 2023, vol. 36, pp. 8634—8652. DOI: 10.52202/075280-0377.
4. Besta M., Blach N., Kubicek A. et al. Graph of Thoughts: Solving Elaborate Problems with Large Language Models, *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024, vol. 38, no. 16, pp. 17682—17690. DOI: 10.1609/aaai.v38i16.29720.
5. Zhou A., Yan K., Shlapentokh-Rothman M. et al. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models, *Proceedings of Machine Learning Research*, 2024, vol. 235, pp. 62138—62160.
6. Harman M. The Current State and Future of Search-Based Software Engineering, *Future of Software Engineering (FOSE 2007)*. Washington, IEEE Computer Society, 2007, pp. 342—357. DOI: 10.1109/FOSE.2007.29.
7. Harman M., Mansouri S. A., Zhang Y. Search-Based Software Engineering: Trends, Techniques and Applications, *ACM Computing Surveys*, 2012, vol. 45, no. 1, article 11. DOI: 10.1145/2379776.2379787.
8. McMinn P. Search-Based Software Test Data Generation: A Survey. *Software Testing, Verification and Reliability*, 2004, vol. 14, no. 2, pp. 105—156. DOI: 10.1002/stvr.294.
9. Antamoshkin O. A. *Software Engineering: Theory and Practice*. Textbook. Krasnoyarsk: Siberian Federal University, 2012, 247 p. (in Russian).
10. Antamoshkin O. A., Antamoshkina O. A., Smirnov N. A. Multi-agent automation system for monitoring, forecasting and managing emergency situations, *IOP Conference Series: Materials Science and Engineering*, 2016, vol. 122, article 012003. DOI: 10.1088/1757-899X/122/1/012003.
11. Sartaj H., Ali Sh., Arcaini P., Arcuri A. Search-Based Software Engineering and AI Foundation Models: Current Landscape and Future Roadmap, *arXiv preprint*, 2025, arXiv:2505.19625. DOI: 10.48550/arXiv.2505.19625.

ИНФОРМАЦИЯ

Продолжается подписка на журнал «Программная инженерия» на первое полугодие 2027 г.

Оформить подписку можно через подписные агентства
или непосредственно в редакции журнала (для юридических лиц).

Подписной индекс по Объединенному каталогу

«Пресса России» — 22765

Возможна подписка

на электронную версию нашего журнала:

ООО «ИВИС»: тел. (495) 777-65-57, 777-65-58; e-mail: sales@ivis.ru;

ГК «УП Урал-Пресс». Для оформления подписки (индекс 013312)
следует обратиться в филиал по месту жительства — <http://ural-press.ru>;

Научная электронная библиотека eLIBRARY.RU;

Электронная библиотека Руконт: <https://rucont.ru>.

Адрес редакции: 107076, Москва, Матросская Тишина, д. 23, с. 2, оф. 45.

Издательство «Новые технологии»,
редакция журнала «Программная инженерия»

Тел.: (499) 270-16-52. E-mail: prin@novtex.ru