УДК 004.415.532.2
DOI

МЕТОД НЕИНВАЗИВНОГО АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА НАСТОЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ ДИНАМИЧЕСКОЙ ИНЪЕКЦИИ КОДА

Голдышев Д. М. ORCID ID 0009-0001-4773-5616,
Фетисов М. В. ORCID ID 0000-0002-4363-7920

Федеральное государственное автономное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)», Москва, Российская Федерация, e-mail: goldyshev.work@mail.ru

В современных программных системах значительную долю регрессионных дефектов составляют ошибки, проявляющиеся на уровне графического интерфейса, при этом автоматизация таких проверок осложняется отсутствием доступа к исходному коду и ограниченной наблюдаемостью состояния интерфейса. Существующие подходы, основанные на визуальном распознавании элементов или использовании интерфейсов доступности, зависят от внешнего представления и могут обеспечивать недостаточную точность идентификации компонентов интерфейса. Целью исследования является разработка и обоснование метода неинвазивного автоматизированного тестирования графического интерфейса настольных приложений, использующего динамическую инъекцию кода для обеспечения наблюдаемости и управляемости интерфейса через внутреннюю объектную модель приложения. В статье рассматриваются механизмы внедрения тестового агента в процесс приложения в операционных системах Linux, macOS и Windows на примере приложений на базе фреймворка Qt, а также подходы к извлечению внутренней объектной модели интерфейса и выполнению действий над элементами, идентифицируемыми и адресуемыми через нее. В результате сформулированы принципы применения динамической инъекции кода в задачах автоматизированного тестирования и определены правила взаимодействия с объектной моделью интерфейса. Показана применимость подхода при построении системы тестирования настольных приложений при отсутствии доступа к исходному коду и невозможности его модификации. Сделан вывод о целесообразности использования динамической инъекции кода для повышения наблюдаемости и управляемости графического интерфейса в задачах автоматизированного тестирования.

Ключевые слова: динамическая инъекция кода, автоматизированное тестирование, объектная модель интерфейса, графический пользовательский интерфейс, фреймворк Qt, неинвазивное тестирование

NON-INVASIVE AUTOMATED TESTING METHOD FOR DESKTOP APPLICATION GRAPHICAL INTERFACES BASED ON DYNAMIC CODE INJECTION

Goldyshev D. M. ORCID ID 0009-0001-4773-5616,
Fetisov M. V. ORCID ID 0000-0002-4363-7920

Federal State Autonomous Educational Institution of Higher Education "Moscow State Technical University named after N. E. Bauman", Moscow, Russian Federation, e-mail: goldyshev.work@mail.ru

In modern software systems, a significant proportion of regression defects consists of errors manifesting at the graphical user interface level, while automation of such testing is complicated by the lack of access to the source code and limited observability of the interface state. Existing approaches based on visual element recognition or the use of accessibility interfaces depend on the external representation and may provide insufficient accuracy in identifying interface components. The aim of this work is to develop and substantiate a method for non-invasive automated testing of desktop application graphical user interfaces, using dynamic code injection to ensure observability and controllability of the interface through the application's internal object model. The article examines mechanisms for injecting a test agent into the application process in Linux, macOS and Windows operating systems using Qt framework-based applications as an example, as well as approaches to extracting the internal object model of the interface and performing actions on elements identified and addressed through it. As a result, the principles of applying dynamic code injection in automated testing tasks have been formulated and the rules for interacting with the interface object model have been defined. The applicability of the approach for building a desktop application testing system in the absence of access to the source code and the impossibility of its modification has been demonstrated. The conclusion is made about the advisability of using dynamic code injection to increase observability and controllability of the graphical user interface in automated testing tasks.

Keywords: dynamic code injection, automated testing, graphical user interface, desktop applications, Qt framework, non-invasive testing

Введение

Автоматизированное регрессионное тестирование графического интерфейса является одним из наиболее трудоемких элементов жизненного цикла программного обеспечения, поскольку ошибки взаимодействия и отображения часто проявляются только на уровне пользовательского интерфейса и зависят от динамического состояния приложения [1]. Для настольных приложений данная проблема усугубляется тем, что поставка программного обеспечения нередко осуществляется в виде бинарных сборок, а доступ к исходному коду тестируемого продукта отсутствует либо его модификация недопустима по организационным, юридическим или технологическим причинам [2]. Дополнительным фактором актуальности является ограниченная доступность специализированных средств автоматизированного тестирования GUI в ряде практических сценариев. В таких условиях использование коммерческих решений может быть затруднено лицензионными, организационными или инфраструктурными ограничениями, что повышает значимость воспроизводимых методов, не требующих модификации тестируемого приложения и допускающих адаптацию под конкретную среду разработки [3, 4].

На практике широко распространены внешние методы автоматизации: визуальное распознавание элементов интерфейса, управление на уровне координат и событий ввода, а также использование интерфейсов доступности [5–7]. Однако общим недостатком таких подходов является зависимость от внешнего представления интерфейса и, как следствие, ограниченная точность идентификации элементов в условиях эволюции пользовательского интерфейса [8, 9]. Дополнительным ограничением становится недостаточная наблюдаемость: внешние средства тестирования оперируют косвенными признаками состояния, тогда как значимая часть логики и параметров интерфейсных объектов остается внутри адресного пространства процесса приложения. Тем самым возникает разрыв между наблюдаемыми извне признаками GUI и фактическим состоянием элементов внутри процесса приложения.

Для повышения наблюдаемости и управляемости графического интерфейса целесообразно использовать подход, при котором тестовая подсистема получает доступ к внутренней объектной модели интерфейса непосредственно в процессе выполнения приложения. В контексте приложений на основе фреймворка Qt такая объектная модель является центральным механизмом организации интерфейса: элементы представлены объектами, образующими иерархию, а их свойства и состояния доступны для интроспекции и управления[1]. Доступ к внутренней объектной модели без модификации исходного кода тестируемого приложения может быть обеспечен путем внедрения тестового агента в процесс приложения посредством динамической инъекции кода, то есть принудительной загрузки дополнительной динамической библиотеки в адресное пространство запускаемого процесса. В данном контексте инъекция рассматривается как легитимный инженерный механизм подключения вспомогательной функциональности во время выполнения, применяемый в инструментах профилирования, отладки и анализа программ.

Цель исследования – разработка и обоснование метода неинвазивного автоматизированного тестирования графического интерфейса настольных приложений на основе динамической инъекции кода, обеспечивающего внедрение тестового агента и взаимодействие с внутренней объектной моделью интерфейса без модификации исходного кода тестируемого приложения.

Материалы и методы исследования

Объектом исследования являются настольные приложения с графическим интерфейсом, реализованным с использованием технологий фреймворка Qt, функционирующие в операционных системах Linux, macOS и Windows. Выбор фреймворка Qt обусловлен его распространенностью в разработке кроссплатформенных настольных приложений, наличием развитой объектной модели интерфейса и доступностью механизмов интроспекции.

Методологическая основа исследования включала анализ документации операционных систем и фреймворка Qt в части механизмов динамической загрузки библиотек и интроспекции объектов, проектирование архитектуры метода и прикладную апробацию результатов. Апробация метода проводилась на версии Qt 5.15 LTS. Экспериментальная проверка внедрения тестового агента и наблюдаемости внутренней объектной модели интерфейса выполнялась на стендовом проекте (доступном по адресу: https://github.com/lildannita/qt-injection), включающем тестовые приложения на Qt Widgets и Qt Quick/QML, внедряемую библиотеку агента и средства запуска для Linux, macOS и Windows.

[1] Qt Meta-Object System. [Электронный ресурс].URL: https://doc.qt.io/archives/qt-5.15/metaobjects.html (дата обращения: 01.01.2026).

Результаты исследования и их обсуждение

Динамическая инъекция кода в рамках настоящей работы рассматривается как инженерный механизм внедрения тестового агента (динамической библиотеки) в адресное пространство процесса тестируемого приложения на этапе запуска или во время выполнения. Внедрение агента обеспечивает выполнение тестовой логики в контексте процесса приложения и тем самым доступ к внутренней объектной модели графического интерфейса без модификации исходного кода. Для приложений на базе фреймворка Qt это принципиально, поскольку элементы интерфейса представлены иерархией объектов (Qt Widgets либо Qt Quick/QML), а их состояния доступны для интроспекции через объектную модель.

В Linux внедрение тестового агента реализуется предзагрузкой динамической библиотеки агента средствами динамического загрузчика (LD_PRELOAD), что обеспечивает подключение агента до загрузки основных зависимостей приложения [10]. В macOS используется DYLD_INSERT_LIBRARIES, при этом системные механизмы защиты ограничивают внедрение для части системных компонентов, однако в контролируемой тестовой среде метод применим к пользовательским Qt-приложениям [11]. Пример запуска с внедрением агента в Linux и macOS приведен в листинге 1.

В Windows отсутствует встроенный механизм предзагрузки, поэтому внедрение агента обычно реализуется созданием процесса в приостановленном состоянии и инъекцией DLL через системные API, после чего выполнение возобновляется [12]. Для Windows существенны условия совпадения разрядности агента и целевого процесса, требования к привилегиям и возможное влияние средств защиты в инфраструктуре тестирования.

Результаты определения и сопоставления механизмов внедрения тестового агента для настольных операционных систем Linux, macOS и Windows представлены в таблице.

```
LD_PRELOAD=/path/to/libagent.so /path/to/application # Linux
DYLD_INSERT_LIBRARIES=/path/to/libagent.dylib /path/to/application # macOS
```

Листинг 1. Пример запуска приложения с внедрением тестового агента в Linux и в macOS
Примечание: составлен авторами на основе источников [10, 11]

Сравнение механизмов динамической инъекции кода

Характеристика	Linux	macOS	Windows
Механизм	LD_PRELOAD	DYLD_INSERT_LIBRARIES	CreateRemoteThread + LoadLibrary
Сложность реализации	Низкая	Низкая	Средняя
Расширение библиотеки	.so	.dylib	.dll
Системные ограничения	setuid/setgid	SIP	UAC, антивирусы

Примечание: составлена авторами на основе полученных данных в ходе исследования.

На основе рассмотренных механизмов внедрения разработана архитектура метода неинвазивного тестирования, включающая внешнюю систему тестирования, тестовый агент внутри процесса приложения и слой взаимодействия с объектной моделью интерфейса. Внешняя система отвечает за выполнение сценариев и формирование отчетности, что соответствует распространенной практике разделения тестового рантайма и тестируемого приложения [13]. Агент обеспечивает наблюдение за объектной моделью, адресацию элементов, выполнение действий и получение состояния для проверок. Критичным требованием является неинвазивность – внедрение агента и обработка событий объектной модели не должны изменять функциональное поведение тестируемого приложения.

Ключевым результатом является способ поддержания актуального представления объектной модели Qt-приложения без модификации исходного кода. Для этого тестовый агент использует внутренний механизм хуков Qt и обрабатывает события добавления и удаления объектов[2]. Этого достаточно для поддержания актуального дерева объ-

[2] Qt source code: qhooks.cpp. [Электронный ресурс]. URL: https://codebrowser.dev/qt5/qtbase/src/corelib/global/qhooks.cpp.html (дата обращения: 04.02.2026).

ектов интерфейса в процессе выполнения. Инъекция в данном контексте выступает как фильтр наблюдения. Через переопределенные обработчики проходит поток событий жизненного цикла объектов. Для обеспечения неинвазивности агент сохраняет адреса исходных обработчиков хуков и вызывает их после собственной обработки события. Это сохраняет семантику работы приложения и совместимость с возможным пользовательским кодом, использующим тот же механизм. Принцип установки хуков и делегирования исходной логике показан на примере обработчиков добавления и удаления объектов, представленном в листинге 2.

```
#include <private/qhooks_p.h>
#include <QCoreApplication>
#include <QMetaObject>
static QHooks::StartupCallback next_startupHook = nullptr;
static QHooks::AddQObjectCallback next_objectAddedHook = nullptr;
static QHooks::RemoveQObjectCallback next_objectRemovedHook = nullptr;
extern "C" Q_DECL_EXPORT void startupHook() {
 // Отложенная инициализация, чтобы не работать с частично готовым QCoreApplication
    QMetaObject::invokeMethod(qApp, []{
        // Установка компонента наблюдения (например, eventFilter на QCoreApplication)
    }, Qt::QueuedConnection); // выполняется после запуска цикла обработки событий
    if (next_startupHook) next_startupHook();
}
extern "C" Q_DECL_EXPORT void objectAddedHook(QObject* obj) {
    // Регистрация объекта (обновление представления объектной модели)
    if (next_objectAddedHook) next_objectAddedHook(obj);
}
extern "C" Q_DECL_EXPORT void objectRemovedHook(QObject* obj)
{
    // Удаление объекта (обновление представления объектной модели)
    if (next_objectRemovedHook) next_objectRemovedHook(obj);
}
static void installHooks()
{
     next_startupHook = reinterpret_cast<QHooks::StartupCallback>(qtHookData[QHooks::Start
up]);
    next_objectAddedHook =
      reinterpret_cast<QHooks::AddQObjectCallback>(qtHookData[QHooks::AddQObject]);
    next_objectRemovedHook =
       reinterpret_cast<QHooks::RemoveQObjectCallback>(qtHookData[QHooks::RemoveQObje
ct]);
    qtHookData[QHooks::Startup] = reinterpret_cast<quintptr>(&startupHook);
    qtHookData[QHooks::AddQObject] = reinterpret_cast<quintptr>(&objectAddedHook);
   qtHookData[QHooks::RemoveQObject] = reinterpret_cast<quintptr>(&objectRemovedHook);
}
Q_COREAPP_STARTUP_FUNCTION(installHooks)
```

Листинг 2. Установка хуков Qt и делегирование исходным обработчикам
Примечание: составлен авторами по результатам данного исследования

Для выполнения тестовых действий и проверок сформулирован способ адресации элементов интерфейса на основе иерархического пути в объектной модели. Путь состоит из идентификаторов узлов от корневого объекта до целевого. Идентификатор узла формируется по приоритету признаков. При наличии задаваемого разработчиком имени объекта оно используется как предпочтительный признак. В противном случае допускается использование других доступных атрибутов элемента. При отсутствии семантических атрибутов применяется резервный механизм, основанный на типе объекта и его позиции среди однотипных элементов на текущем уровне иерархии. Такой порядок обеспечивает применимость метода без обязательного изменения тестируемого приложения и позволяет получать стабильные адреса для типовых интерфейсных структур.

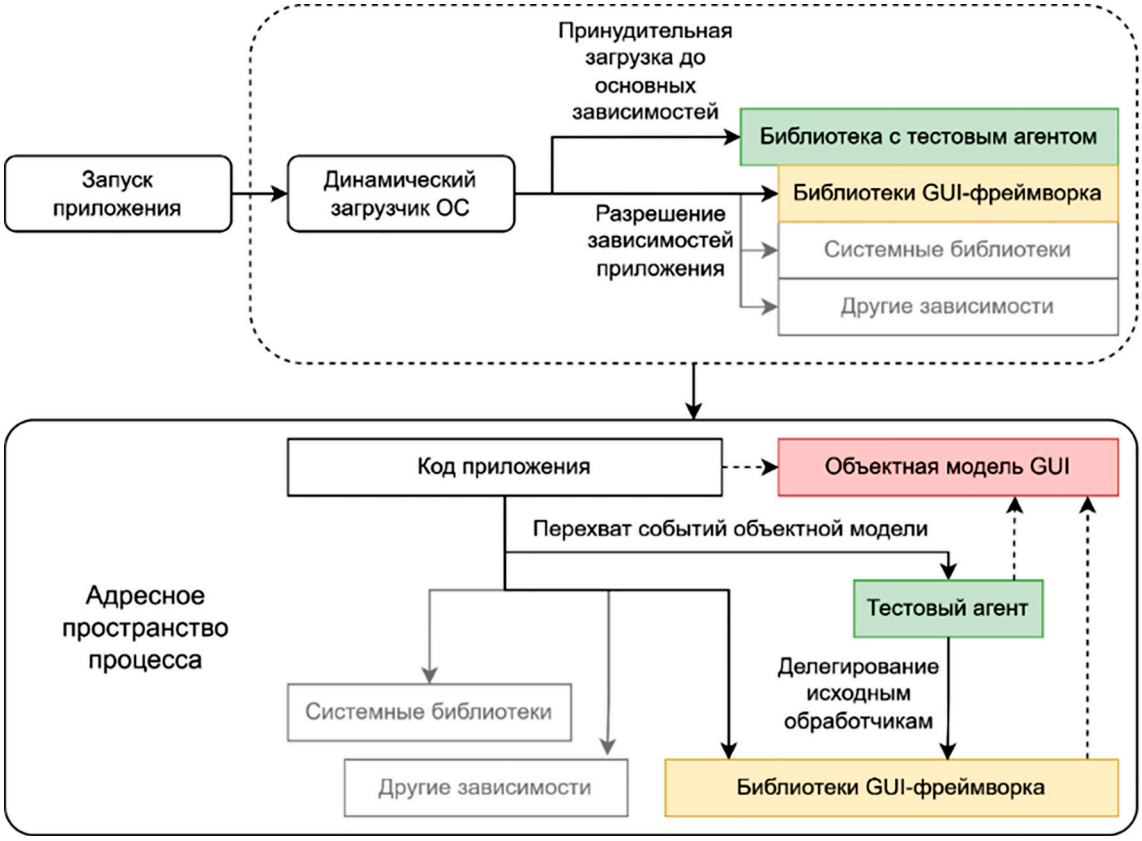

Рис. 1. Схема работы тестового агента в адресном пространстве процесса и доступа к объектной модели GUI
Примечание: составлен авторами по результатам данного исследования

На рис. 1 показано, как внедренный тестовый агент располагается в адресном пространстве процесса и встраивается в цепочку обработки событий объектной модели графического интерфейса. Показанный ранее принцип перехвата событий реализуется таким образом, что агент получает возможность регистрировать изменения в объектной модели и выполнять прикладную тестовую логику, сохраняя при этом штатное поведение приложения за счет делегирования исходным обработчикам. Диаграмма подчеркивает, что агент является частью процесса наряду с библиотеками GUI-фреймворка и системными зависимостями, а взаимодействие с объектной моделью осуществляется через механизмы самого фреймворка. Это обеспечивает наблюдаемость структуры интерфейса и возможность выполнения тестовых действий на уровне внутренних объектов без вмешательства в код приложения.

Для экспериментальной проверки метода разработан стендовый проект, включающий два тестовых приложения (Qt Widgets и Qt Quick/QML), внедряемую библиотеку тестового агента на C++ и средства запуска, учитывающие различия механизмов внедрения в Linux, macOS и Windows. В рамках эксперимента агент перехватывает события пользовательского взаимодействия с элементами интерфейса и выводит в консоль тип события, реальный класс объекта, заданное имя (если есть) и иерархический путь элемента в объектной модели. Это подтверждает наблюдаемость внутренней объектной модели интерфейса во время выполнения без модификации исходного кода приложения. Пример результата внедрения и диагностического вывода в среде Windows приведен на рис. 2, 3.

Следует отметить, что динамическая инъекция кода используется в ряде инструментов программной инженерии. Valgrind применяет внедрение для динамического анализа и профилирования [14]. AddressSanitizer использует подключение проверок времени выполнения в тестовой среде [15]. GammaRay применяет инъекцию для интроспекции Qt-приложений и анализа объектной модели[3].

[3] GammaRay – Qt application inspection tool. [Электронный ресурс]. URL: https://kdab.github.io/GammaRay (дата обращения: 07.01.2026).

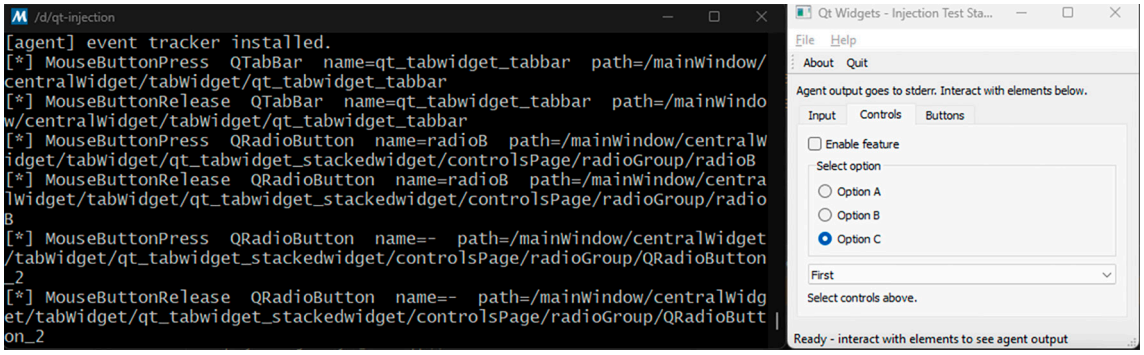

Рис. 2. Диагностический вывод тестового агента при взаимодействии с элементами Qt Widgets (Windows)
Примечание: составлен авторами по результатам данного исследования

Рис. 3. Диагностический вывод тестового агента при взаимодействии с элементами Qt Quick/QML (Windows)
Примечание: составлен авторами по результатам данного исследования

В отличие от перечисленных инструментов, в настоящей работе инъекция формализована как элемент воспроизводимого метода автоматизированного тестирования графического интерфейса. Метод включает внедрение агента, поддержание дерева объектов, адресацию элементов и выполнение действий и проверок по фактическому состоянию объектов.

Коммерческая система Squish демонстрирует практическую реализуемость доступа к объектной модели Qt для задач тестирования без модификации исходного кода[4]. При этом закрытость реализации ограничивает воспроизводимость и адаптацию подхода, что обосновывает необходимость открытого описания метода и его кроссплатформенных аспектов.

Полученные решения доведены до практического применения в составе разработанной системы автоматизированного тестирования QtAda (доступной по адресу https://github.com/lildannita/QtAda) и применяются для регрессионных проверок GUI в контуре CI/CD на платформе Linux. В реализации QtAda обеспечены внедрение тестового агента, поддержание актуального представления объектной модели интерфейса, адресация элементов и выполнение действий и проверок по фактическому состоянию объектов.

К ограничениям метода относятся необходимость динамического связывания приложения с библиотеками Qt, поскольку статическая сборка ограничивает применимость инъекции. Координатные действия рассматриваются как вспомогательные, а основной режим взаимодействия реализуется через операции над объектами внутренней объектной модели интерфейса.

Заключение

В результате исследования разработан метод неинвазивного автоматизированного тестирования графического интерфейса настольных приложений на основе динамической инъекции кода. Метод обеспечивает внедрение тестового агента в процесс

[4] Squish: Hooking into Subprocesses. [Электронный ресурс]. URL: https://wiki.qt.io/Squish/Hooking_into_Subprocesses (дата обращения: 10.01.2026).

приложения без модификации исходного кода и доступ к внутренней объектной модели интерфейса для выполнения действий и проверок по фактическому состоянию элементов.

Систематизированы механизмы инъекции в операционных системах Linux, macOS и Windows. Разработана архитектура метода, включающая внешнюю систему тестирования, тестовый агент и слой взаимодействия с объектной моделью. Сформулированы принципы адресации элементов на основе иерархического пути в объектной модели.

Применимость метода подтверждена апробацией в составе системы автоматизированного тестирования QtAda. Открытое описание метода позволяет воспроизвести и адаптировать подход для задач тестирования Qt-приложений.

Список литературы

1. Nass M., Alégroth E., Feldt R. Why many challenges with GUI test automation (will) remain // Information and Software Technology. 2021. Vol. 138. Art. 106625. DOI: 10.1016/j.infsof.2021.106625.

2. Banerjee I., Nguyen B., Garousi V., Memon A. Graphical user interface (GUI) testing: Systematic mapping and repository // Information and Software Technology. 2013. Vol. 55. Is. 10. P. 1679–1694. DOI: 10.1016/j.infsof.2013.03.004.

3. Alégroth E., Feldt R., Kolström P. Maintenance of Automated Test Suites in Industry: An Empirical Study on Visual GUI Testing // Information and Software Technology. 2016. Vol. 73. P. 66–80. DOI: 10.1016/j.infsof.2016.01.012.

4. Chahim H., Vos T., Baars A. Scriptless Testing at the GUI Level in an Industrial Setting // Lecture Research Challenges in Information Science. Springer, Cham. 2020. P. 267–284. DOI: 10.1007/978-3-030-50316-1_16.

5. Nie L., Said K. S., Ma L., Zheng Y., Zhao Y. A Systematic Mapping Study for Graphical User Interface Testing on Mobile Apps // IET Software. 2023. Vol. 17. Is. 3. P. 249–267. DOI: 10.1049/sfw2.12123.

6. Yeh T., Chang T.-H., Miller R. C. Sikuli: Using GUI Screenshots for Search and Automation // Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST ‘09). ACM, New York. 2009. P. 183–192. DOI: 10.1145/1622176.1622213. ISBN 978-1-60558-745-5.

7. Yu S., Fang C., Li X., Ling Y., Chen Z., Su Z. Effective, Platform-Independent GUI Testing via Image Embedding and Reinforcement Learning // ACM Transactions on Software Engineering and Methodology. ACM. 2024. DOI: 10.1145/3674728.

8. Vos T. E. J., Aho P., Ricós F. P., Valdes O. R., Mulders A. Testar: Scriptless Testing Through Graphical User Interface // Software Testing, Verification and Reliability. 2021. Vol. 31. Is. 3. Art. e1771. DOI: 10.1002/stvr.1771.

9. Spinak J. Model-based GUI automation // Software and Systems Modeling. Springer. 2025. DOI: 10.1007/s10270-025-01319-9.

10. Gómez Moreno J. M., Moutafis V., Dionysiou A., Kuipers F., Smaragdakis G., Coppens B., Voulimeneas A. Clair Obscur: The Light and Shadow of System Call Interposition – From Pitfalls to Solutions with K23 // Proceedings of the 26th International Middleware Conference (Middleware ’25). ACM, New York. 2025. P. 241–255. DOI: 10.1145/3721462.3770772.

11. Voss M., Asenjo R., Reinders J. Scalable Memory Allocation // Pro TBB: C++ Parallel Programming with Threading Building Blocks. Springer, Cham. 2019. P. 207–231. DOI: 10.1007/978-1-4842-4398-5_7.

12. Hunt G., Brubacher D. Detours: Binary Interception of Win32 Functions // Proceedings of the 3rd USENIX Windows NT Symposium. USENIX Association. 1999. DOI: 10.5555/1268427.1268441.

13. Buender H., Kuchen H. A Model-Driven Approach for Behavior-Driven GUI Testing // Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC ’19). ACM, New York. 2019. P. 1742–1751. DOI: 10.1145/3297280.3297450.

14. Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation // ACM SIGPLAN Notices. 2007. Vol. 42. № 6. P. 89–100. DOI: 10.1145/1273442.1250746.

15. Serebryany K., Bruening D., Potapenko A., Vyukov D. AddressSanitizer: A Fast Address Sanity Checker // Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC ’12). 2012. P. 309–318. DOI: 10.5555/2342821.2342849.

Конфликт интересов: Авторы заявляют об отсутствии конфликта интересов.

Conflict of interest: The authors declare that there is no conflict of interest.