УДК 004.415.53
DOI 10.17513/snt.40817



# МЕТОД АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ANDROID-ПРИЛОЖЕНИЙ БЕЗ ИЗМЕНЕНИЯ ИСХОДНОГО КОДА С ВНЕДРЕНИЕМ ТЕСТОВОГО АГЕНТА

**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*

Автоматизация тестирования графического интерфейса Android-приложений затрудняется, если тестируемый продукт поставляется в виде готового пакета, а изменение его исходного кода недопустимо. Внешние средства автоматизации работают только с видимым представлением и не дают доступа к внутренним объектам интерфейса приложения. Целью работы является расширение приёма динамической инъекции кода на платформу Android при сохранении неинвазивности: внедрение тестового агента в процесс приложения, организация канала связи с внешней системой тестирования и передача параметров сеанса без изменения исходного кода и манифеста приложения. Методологическую основу составили анализ механизмов динамической загрузки разделяемых библиотек, изучение средств проброса соединений, проектирование архитектуры метода и его прикладная апробация на эмуляторе Android для приложений на основе фреймворка Qt. В результате определены действия по формированию тестового экземпляра приложения, способ загрузки агента вместе с приложением и схема обмена данными с внешней системой тестирования без изменения состава компонентов и разрешений приложения. Сформулированы условия неинвазивности через сохранение публичного контракта приложения и семантики его наблюдаемого поведения, определены границы применимости метода. Полученные результаты подтверждают возможность тестирования графического интерфейса Android-приложений на основе фреймворка Qt с доступом к объектной модели интерфейса без изменения исходного кода тестируемого продукта.

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

# METHOD FOR AUTOMATED GRAPHICAL INTERFACE TESTING OF ANDROID APPLICATIONS WITHOUT SOURCE CODE MODIFICATION THROUGH TEST AGENT 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 «Bauman Moscow State Technical University», Moscow, Russian Federation, e-mail: goldyshev.work@mail.ru*

Automating graphical interface testing of Android applications is hindered when the application under test is shipped as a prebuilt package and modifying its source code is not permitted. External automation tools operate only on the visible representation and do not provide access to the internal objects of the application interface. The aim of the work is to extend dynamic code injection to the Android platform while preserving non-invasiveness: injecting the test agent into the application process, organizing a communication channel with the external testing system, and passing session parameters without modifying the source code or the application manifest. The methodological basis comprised analysis of dynamic loading mechanisms for shared libraries, study of connection-forwarding facilities, design of the method architecture, and its applied evaluation on an Android emulator for applications based on the Qt framework. As a result, the steps for preparing a test instance, loading the agent together with the application, and the data exchange scheme with the external testing system have been defined without changing the components or permissions of the application. The conditions of non-invasiveness, based on preserving the public contract of the application and the semantics of its observable behavior, were formulated, and the applicability boundaries of the method were defined. The obtained results confirm the feasibility of testing the graphical interface of Android applications based on the Qt framework with access to the interface object model without modification of the source code of the application under test.

**Keywords: dynamic code injection, automated testing, graphical user interface, interface object model, Qt framework, non-invasive testing, Android**

## Введение

Автоматизированное регрессионное тестирование графического интерфейса мобильных приложений приобретает особую значимость по мере роста сложности пользовательских сценариев и частоты выпуска обновлений на платформе Android [1-3]. Ошибки взаимодействия и отображения нередко обусловлены динамическим состоянием приложения и проявляются исключительно на уровне пользовательского интерфейса [4; 5], что повышает требования к точности и устойчивости средств автоматизации. Поставка приложений в виде упакованных артефактов формата APK дополнительно усложняет задачу: исходный код тестируемого продукта зачастую недоступен либо его модификация недопустима по организационным или юридическим причинам, а формат APK, в отличие от настольных бинарных артефактов, дополнительно регламентирован требованиями подписи и метаданных уровня операционной системы.

Существующие подходы к автоматизации тестирования Android-приложений включают визуальное распознавание элементов интерфейса, средства уровня операционной системы и решения, опирающиеся на интерфейсы доступности или скриншоты [6-8]. Внешние подходы зависят от внешнего представления интерфейса и характеризуются недостаточной точностью идентификации элементов в условиях эволюции графического интерфейса [9-11]. Решения, требующие интеграции в проект тестируемого приложения, противоречат сценарию работы с готовым APK без доступа к исходному коду [12-14].

В ранее опубликованной работе авторами был сформулирован метод неинвазивного автоматизированного тестирования графического интерфейса настольных Qt-приложений [15]. Данный метод был рассмотрен применительно к настольным операционным системам, где внедрение тестового агента может выполняться с использованием штатных механизмов предзагрузки или принудительной загрузки разделяемых библиотек. На платформе Android такие механизмы не имеют прямого аналога, поскольку приложение поставляется в виде APK-пакета, а его компоненты, разрешения и параметры запуска фиксируются в манифесте AndroidManifest.xml – декларативном описании приложения. Поэтому перенос подхода на Android требует решения трёх связанных задач: внедрения тестового агента в адресное пространство процесса приложения без модификации исходного кода и декларативного описания приложения; организации канала связи между агентом и внешней системой тестирования без добавления новых Android-разрешений; передачи параметров сеанса агенту в условиях отсутствия привычной для настольных приложений командной строки и управляемого набора переменных окружения процесса.

Под неинвазивностью в настоящей работе понимается сохранение публичного контракта приложения (имени пакета, набора компонентов, заявленных разрешений) и семантики его поведения. Соответствие этому требованию принципиально для применимости метода к Android-приложениям рассматриваемого класса, поставляемым в виде готового APK, поскольку нарушение хотя бы одного из элементов публичного контракта может привести к расхождению наблюдаемого поведения с поведением исходного приложения.

**Цель исследования** – расширение приёма динамической инъекции кода на платформу Android при сохранении неинвазивности: внедрение тестового агента в процесс приложения, организация канала связи с внешней системой тестирования и передача параметров сеанса без правки исходного кода и манифеста.

## Материалы и методы исследования

Объектом исследования являются Android-приложения с графическим интерфейсом, реализованным на основе фреймворка Qt (Qt Widgets либо Qt Quick/QML), поставляемые в виде упакованных артефактов формата APK. Выбор Qt обусловлен распространённостью этого фреймворка для создания кроссплатформенных приложений и развитой объектной моделью интерфейса [15].

Методологическая основа исследования включала анализ документации Android NDK и инструментов сборки в части механизмов динамической загрузки разделяемых библиотек и формата APK, изучение механизма проброса соединений средствами ADB, проектирование архитектуры метода и его прикладную апробацию. Апробация метода проводилась на наборе тестовых Android-приложений, разработанных с использованием Qt 5.15, 6.5 и 6.8. В состав набора входили приложения, реализующие графический интерфейс с применением Qt Widgets и Qt Quick/QML. Экспериментальная проверка была направлена на подтверждение применимости метода к различным версиям Qt и двум основным технологиям построения интерфейса в рамках единой схемы внедрения тестового агента. Проверка выполнялась на эмуляторе Android.

**Результаты исследования и их обсуждение**

На платформе Android подготовка к динамической инъекции кода выполняется на этапе формирования тестового экземпляра APK и заключается в дополнении его файловой структуры тестовым агентом без изменения логики самого приложения, а непосредственная загрузка агента происходит с помощью штатного механизма динамической компоновки Android. По аналогии с механизмами предзагрузки разделяемых библиотек в Linux и macOS [15] предлагаемый подход использует штатный механизм разрешения зависимостей при загрузке нативных библиотек.

Подготовительный этап включает анализ исходного APK как контейнера приложения: определяется целевая архитектура процессора, состав размещённых в пакете нативных разделяемых библиотек и версия фреймворка Qt, с которой было собрано тестируемое приложение. Получение мажорной и минорной версий Qt необходимо для выбора сборки тестового агента, совместимой с приложением на уровне бинарного интерфейса: несовпадение версии может привести к ошибкам загрузки библиотеки агента либо к некорректной работе при обращении к объектной модели интерфейса. Мажорная версия определяется по именам базовых библиотек Qt, а минорная – по метаданным, содержащимся в нативной библиотеке ядра Qt.

После выбора совместимого агента формируется временный тестовый экземпляр APK: архив приложения распаковывается, в область нативных библиотек выбранной архитектуры копируется библиотека агента и только отсутствующие зависимости, необходимые для её загрузки, а в динамическую секцию основной нативной библиотеки приложения с помощью patchelf[1] добавляется запись DT_NEEDED с именем библиотеки агента. Затем тестовый экземпляр APK заново упаковывается без служебных данных исходной подписи, выравнивается утилитой zipalign[2] в соответствии с требованиями Android и подписывается тестовым ключом с помощью apksigner[3]. Принципиальной особенностью является то, что файл AndroidManifest.xml на этой цепочке не модифицируется: имя пакета, набор компонентов приложения, объявленные разрешения и иные элементы публичного контракта остаются неизменными. В обобщённом виде алгоритм формирования тестового экземпляра APK представлен в листинге 1.

Из перечисленных операций единственная содержательная модификация бинарных артефактов – добавление одной записи DT_NEEDED в динамическую секцию основной разделяемой библиотеки приложения. Машинный код основной нативной библиотеки не изменяется: преобразование затрагивает только её служебную динамическую секцию, в которую добавляется запись о зависимости от агента. Остальные прикладные компоненты исходного APK не модифицируются; из тестовой копии исключаются только служебные данные исходной подписи, поскольку после переупаковки они становятся недействительными. Механизм добавления записи DT_NEEDED по назначению сопоставим с предзагрузкой через LD_PRELOAD в Linux: в обоих случаях обеспечивается загрузка дополнительной разделяемой библиотеки в адресное пространство процесса при запуске приложения. В результате агент загружается на раннем этапе инициализации нативной части приложения, до начала основной работы Qt-приложения, что создаёт условия для регистрации хуков объектной модели интерфейса.

Установление связи между агентом, работающим внутри процесса Android-приложения, и внешней системой тестирования осложняется тем, что использование сетевых TCP-сокетов на стороне приложения требует объявления разрешения android.permission.INTERNET в манифесте приложения. Добавление такого разрешения изменяло бы публичный контракт приложения и нарушало бы требование неинвазивности. Поэтому в качестве транспортного механизма на стороне устройства используется Unix domain socket в abstract namespace[4], то есть локальный сокет семейства AF_UNIX, адресуемый не файловым путём, а именем в пространстве имён ядра. Такой сокет не создаёт объекта в файловой системе и не относится к сетевому семейству AF_INET, вследствие чего его использование не требует ни дополнительных прав доступа к файлам, ни объявления сетевого разрешения Android.

Связь с внешней системой тестирования устанавливается с использованием штатного механизма проброса соединений Android Debug Bridge[5].

[1] patchelf – A small utility to modify the dynamic linker and RPATH of ELF executables. URL: https://github.com/NixOS/patchelf (дата обращения: 02.04.2026).

[2] Android Developers. zipalign. URL: https://developer.android.com/tools/zipalign (дата обращения: 02.04.2026).

[3] Android Developers. apksigner. URL: https://developer.android.com/tools/apksigner (дата обращения: 02.04.2026).

[4] Linux Programmer's Manual: unix(7) – sockets for local interprocess communication. URL: https://man7.org/linux/man-pages/man7/unix.7.html (дата обращения: 11.04.2026).

[5] Android Developers. Android Debug Bridge (adb) reference: forward command. URL: https://developer.android.com/tools/adb#forwardports (дата обращения: 16.04.2026).

Входные данные:
A – исходный APK;
B – целевая ABI;
L_agent – разделяемая библиотека тестового агента;
L_main – основная нативная библиотека приложения;
D_agent – множество зависимостей, необходимых для загрузки агента;
D_app – множество нативных библиотек, присутствующих в исходном APK;
K – тестовый ключ подписи.
Выходные данные:
A_test – подписанный тестовый APK.
1. Создать временный рабочий каталог W.
2. Распаковать A в W.
3. Удалить из W служебные данные исходной подписи.
4. Поместить L_agent в каталог нативных библиотек для ABI B.
5. Для каждой зависимости d ∈ D_agent: если d ∉ D_app, то скопировать d из комплекта Qt for Android, использованного при сборке исходного приложения.
6. Добавить в динамическую секцию L_main запись DT_NEEDED с именем L_agent:
   patchelf --add-needed name(L_agent) L_main.
7. Упаковать содержимое W в неподписанный APK A_unsigned.
8. Выполнить выравнивание A_unsigned:
   zipalign -f 4 A_unsigned A_aligned.
9. Подписать A_aligned тестовым ключом K:
   apksigner sign --ks K --out A_test A_aligned.
10. Вернуть A_test.

*Листинг 1. Алгоритм формирования APK с внедрённым тестовым агентом*
*Примечание: составлено авторами по результатам данного исследования*

```
# Создание локального TCP-порта на управляющей машине
# и перенаправление подключений к abstract socket на Android-устройстве.
adb forward tcp:<host_port> localabstract:<abstract_socket_name>
# Запуск Activity тестируемого приложения.
# Имя abstract socket передаётся через Intent extras, чтобы агент создал и прослушивал точку,
# соответствующую пробросу, заранее настроенному управляющим процессом.
adb shell am start -n <package_name>/<activity_name> \
   --es <ipc_name_key> <abstract_socket_name>
```

*Листинг 2. Обобщённая схема проброса соединения*
*и запуска Android-приложения с передачей имени канала связи*
*Примечание: составлено авторами по результатам данного исследования.*

```
// Создание локального Unix domain socket внутри процесса Android-приложения.
sockaddr_un address;
int socketFd = socket(AF_UNIX, SOCK_STREAM, 0);
std::memset(&address, 0, sizeof(address));
address.sun_family = AF_UNIX;
// Первый байт sun_path равен ‘\0’, поэтому имя сокета
// интерпретируется как имя в abstract namespace, а не как путь в ФС.
std::memcpy(address.sun_path + 1, socketName, socketNameLength);
socklen_t addressLength = offsetof(sockaddr_un, sun_path) + 1 + socketNameLength;
// Агент создаёт конечную точку, к которой ADB перенаправит соединение.
bind(socketFd, reinterpret_cast<sockaddr*>(&address), addressLength);
// После listen агент может принять подключение управляющего процесса.
listen(socketFd, backlog);
```

*Листинг 3. Ключевые операции создания Unix domain socket*
*в abstract namespace на стороне агента*
*Примечание: составлено авторами по результатам данного исследования.*

На управляющей машине создаётся локальный TCP-порт, а удалённой конечной точкой проброса указывается Unix domain socket в abstract namespace на Android-устройстве. Поскольку механизм adb forward не создаёт такой сокет самостоятельно, он должен быть предварительно создан процессом, работающим в Android-среде. В предлагаемом методе эту функцию выполняет тестовый агент, находящийся внутри процесса приложения: он создаёт и прослушивает именованный abstract socket, после чего управляющий процесс подключается к локальному TCP-порту на хост-машине, перенаправленному средствами ADB к этому сокету. После принятия соединения агентом формируется двусторонний IPC-канал, в рамках которого дальнейшее взаимодействие агента и управляющего процесса не зависит от исходного разделения на слушающую и подключающуюся стороны. Такая организация связи не требует повышения привилегий приложения на устройстве и не изменяет набор его Android-разрешений. Обобщённая последовательность настройки канала связи и запуска приложения приведена в листинге 2, а ключевые операции создания abstract socket на стороне агента – в листинге 3.

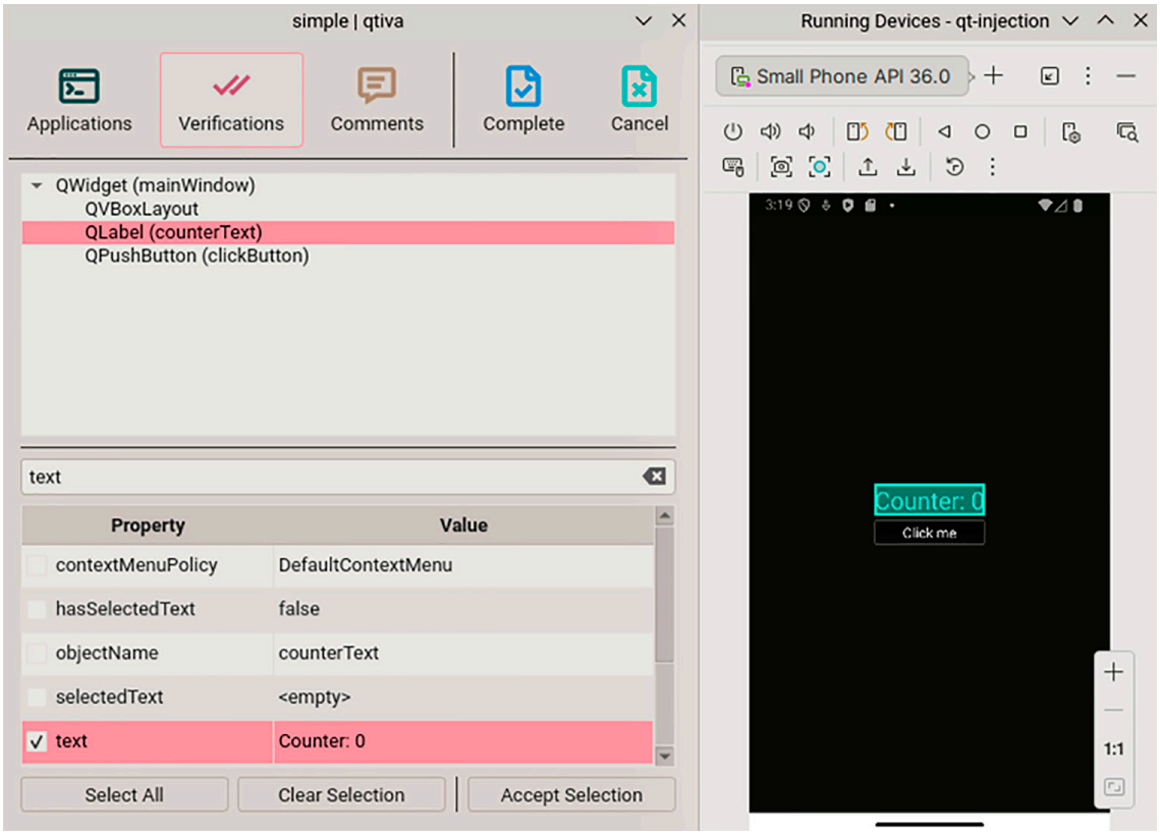
Передача служебных параметров тестового сеанса осуществляется через механизм Intent extras, то есть через дополнительные пары «ключ – значение», включаемые в объект Intent при запуске Android Activity. В предлагаемой схеме таким образом передаётся, в частности, имя Unix domain socket в abstract namespace, необходимое агенту для создания конечной точки связи с управляющим процессом. Команда adb shell am start поддерживает передачу строковых параметров с помощью аргумента --es, что позволяет задать эти данные на этапе запуска приложения без изменения манифеста и без использования переменных окружения процесса. После загрузки агент считывает переданные значения через интерфейс Java Native Interface, обращаясь к методам объекта Intent, связанного с текущей Activity.

Особенность данного способа состоит в том, что агент может быть загружен раньше, чем Qt-слой Android-приложения завершит привязку текущей Activity к нативному окружению. Поэтому чтение параметров сеанса выполняется не однократно, а с использованием ограниченного числа отложенных попыток с небольшой задержкой между ними. Такая схема учитывает жизненный цикл Qt for Android и не требует синхронизации с пользовательским кодом приложения, что важно для сохранения неинвазивности метода и его применимости к приложениям с различной структурой инициализации.

Поддержание актуального представления объектной модели интерфейса внутри процесса осуществляется тем же способом, что и для настольных приложений [15]: тестовый агент устанавливает обработчики хуков Startup, AddQObject и RemoveQObject через интерфейс qtHookData, сохраняя адреса исходных обработчиков и делегируя им выполнение после собственной обработки. Это сохраняет семантику работы приложения и совместимость с возможным пользовательским кодом, использующим тот же механизм. Адресация элементов интерфейса осуществляется по иерархическому пути в объектной модели, формируемому по приоритету признаков элемента: заданное имя объекта, иные семантические атрибуты, тип объекта и позиция среди однотипных элементов на текущем уровне иерархии.

Для прикладной апробации предложенный метод был реализован в экспериментальном прототипе системы автоматизированного тестирования Qt-приложений qtiva. В рамках настоящей работы qtiva рассматривается не как предмет самостоятельного анализа, а как экспериментальная реализация предложенного метода, позволяющая проверить его применимость в полном тестовом сеансе. Для каждой конфигурации из описанного выше набора проверялись подготовка тестового APK, запуск приложения на эмуляторе Android, загрузка агента в адресное пространство приложения, передача параметров сеанса, установление IPC-канала и получение сведений об объектной модели графического интерфейса.

На рисунке приведён репрезентативный пример одного из тестовых сеансов для приложения на Qt 5.15 с интерфейсом Qt Widgets: в эмуляторе Android выполняется тестируемое приложение, а в отдельном окне управляющего процесса отображается дерево объектов его интерфейса, полученное агентом изнутри процесса приложения. Наличие такого дерева демонстрирует достижение основного результата апробации: внешний процесс получает сведения о внутренней объектной модели Android-приложения без модификации исходного кода и манифеста. При этом рисунок иллюстрирует один экспериментальный прогон, тогда как проверка работоспособности метода выполнялась для всего набора тестовых приложений.




*Отображение дерева объектов интерфейса Android-приложения в управляющей панели при записи тестового сценария*
*Примечание: получено авторами в ходе экспериментальной апробации метода*

К ограничениям метода относится необходимость согласования мажорной и минорной версий Qt, использованных при сборке тестируемого приложения и тестового агента. Отдельным ограничением является формирование производного APK, подписанного тестовым ключом: в условиях контролируемого тестового стенда это допустимо, однако для приложений, поведение которых зависит от исходной подписи, требуется дополнительный режим проверки либо отдельное обоснование применимости метода.

## Заключение

В результате исследования разработан метод неинвазивного автоматизированного тестирования графического интерфейса Android-приложений на основе динамической инъекции кода. Метод обеспечивает внедрение тестового агента в процесс приложения без модификации исходного кода и манифеста, передачу параметров сеанса через механизм Intent extras и установление связи между агентом и внешней системой тестирования с использованием Unix domain socket в abstract namespace и штатных средств проброса соединений ADB.

Систематизированы операции цепочки переупаковки APK, обеспечивающие добавление тестового агента в адресное пространство процесса приложения через механизм динамического связывания операционной системы. Сформулированы условия неинвазивности, заключающиеся в сохранении публичного контракта приложения и семантики его поведения. Применимость метода подтверждена апробацией на эмуляторе Android для тестовых приложений на Qt Widgets и Qt Quick/QML.

## Список литературы

1. Mahmud T., Che M., Ngu A., Yang G. Why Android app testing falls short: empirical insights from open-source projects and a practitioner survey // Empirical Software Engineering. 2025. Vol. 30. No. 6. Art. 163. DOI: 10.1007/s10664-025-10726-x.

2. Linares-Vásquez M., Moran K., Poshyvanyk D. Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing // IEEE International Conference on Software Maintenance and Evolution (ICSME '17). 2017. P. 399–410. DOI: 10.1109/ICSME.2017.27.

3. Kong P., Li L., Gao J., Liu K., Bissyandé T.F., Klein J. Automated Testing of Android Apps: A Systematic Literature

Review // IEEE Transactions on Reliability. 2019. Vol. 68. No. 1. P. 45–66. DOI: 10.1109/TR.2018.2865733.

4. Kousar A., Khan S. U. R., Mashkoor A., Iqbal J. A Systematic Literature Review on Graphical User Interface Testing Through Software Patterns // IET Software. 2025. Vol. 2025. No. 1. Art. 9140693. DOI: 10.1049/sfw2/9140693.

5. Deshmukh P. S., Date S. S., Mahalle P. N., Barot J. Automated GUI Testing for Enhancing User Experience (UX): A Survey of the State of the Art // ICT Systems and Sustainability. Springer, Singapore. 2023. P. 619–628. DOI: 10.1007/978-981-99-5652-4_55.

6. 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. No. 3. P. 249–267. DOI: 10.1049/sfw2.12123.

7. Hu C., Neamtiu I. Automating GUI testing for Android applications // Proceedings of the 6th International Workshop on Automation of Software Test (AST '11). ACM, New York. 2011. P. 77–83. DOI: 10.1145/1982595.1982612.

8. Choudhary S. R., Gorla A., Orso A. Automated Test Input Generation for Android: Are We There Yet? // Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15). IEEE. 2015. P. 429–440. DOI: 10.1109/ASE.2015.89.

9. 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. 2024. Vol. 33. No. 7. P. 175:1–175:27. DOI: 10.1145/3674728.

10. 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. No. 3. Art. e1771. DOI: 10.1002/stvr.1771.

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

12. Gu T., Sun C., Ma X., Cao C., Xu C., Yao Y., Zhang Q., Lu J., Su Z. Practical GUI Testing of Android Applications via Model Abstraction and Refinement // Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE '19). IEEE. 2019. P. 269–280. DOI: 10.1109/ICSE.2019.00042.

13. Su T., Meng G., Chen Y., Wu K., Yang W., Yao Y., Pu G., Liu Y., Su Z. Guided, stochastic model-based GUI testing of Android apps // Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). ACM. 2017. P. 245–256. DOI: 10.1145/3106237.3106298.

14. Moran K., Linares-Vásquez M., Bernal-Cárdenas C., Vendome C., Poshyvanyk D. CrashScope: A Practical Tool for Automated Testing of Android Applications // Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C '17). IEEE. 2017. P. 15–18. DOI: 10.1109/ICSE-C.2017.16.

15. Голдышев Д. М., Фетисов М. В. Метод неинвазивного автоматизированного тестирования графического интерфейса настольных приложений на основе динамической инъекции кода // Современные наукоёмкие технологии. 2026. № 3. С. 14–20. DOI: 10.17513/snt.40700.

**Конфликт интересов:** Авторы заявляют об отсутствии конфликта интересов.

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

**Финансирование:** Авторы заявляют об отсутствии внешнего финансирования.

**Financing:** The research was performed without external funding.