
ФЕДЕРАЛЬНОЕ АГЕНТСТВО
ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ



**НАЦИОНАЛЬНЫЙ
СТАНДАРТ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

ГОСТ Р
*(проект,
первая редакция)*

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**Технологии динамического анализа программного
обеспечения. Общие требования**

Настоящий проект стандарта не подлежит применению до его утверждения

Москва

202_

Предисловие

1 РАЗРАБОТАН Федеральной службой по техническому и экспортному контролю (ФСТЭК России), Федеральным государственным бюджетным учреждением науки Институт системного программирования им. В.П. Иванникова Российской академии наук (ИСП РАН)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 362 «Защита информации»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от ____ 20__ № _____

4 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок - в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования - на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.rst.gov.ru)

© Оформление. ФГБУ «Институт стандартизации», 202__

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

Введение.....	
1 Область применения.....	
2 Нормативные ссылки.....	
3 Термины, определения и сокращения.....	
4 Общие положения.....	
5 Общие требования к организации динамического анализа.....	
6 Общие требования к внедрению технологий динамического анализа.....	
7 Общие требования к регулярному выполнению динамического анализа.....	
8 Описание и требования к технологиям динамического анализа.....	
8.1 Тестирование на основе требований: функциональное тестирование.....	
8.2 Тестирование на основе требований: нефункциональное тестирование	
8.3 Фаззинг-тестирование.....	
8.4 Динамический анализ помеченных данных.....	
8.5 Тестирование на отказ.....	
8.6 Тестирование на проникновение.....	
8.7 Динамическая верификация.....	
8.8 Вспомогательные методы динамического анализа.....	
9 Требования к инструментам динамического анализа.....	
10 Требования к специалистам.....	
Приложение А (рекомендуемое) Применение технологий динамического анализа в процессах разработки безопасного программного обеспечения, определенных в ГОСТ Р 56939-2024, с учетом уровней тестирования..	
Приложение Б (справочное) Диаграмма процесса внедрения технологий динамического анализа.....	
Приложение В (справочное) Диаграмма процесса выполнения регулярного динамического анализа.....	

Введение

Безопасность программного обеспечения достигается путем реализации совокупности процессов, среди которых особое значение имеют процессы тестирования программного обеспечения в режиме функционирования, то есть в динамике.

Тестирование программного обеспечения в динамике может обеспечиваться с применением множества технологий и подходов, при этом в контексте обеспечения безопасности ключевое значение имеют технологии, позволяющие выявлять недостатки программного обеспечения, которые могут привести к реализации угроз информационной безопасности, некорректному функционированию, ошибкам в поведении и отказам в обслуживании информационных систем, в составе которых функционирует программное обеспечение.

Вероятность таких угроз возрастает вследствие использования технологий искусственного интеллекта для автоматизированного сбора и интеллектуального анализа информации о программном обеспечении (в том числе его функциональных возможностях и логике их реализации, интерфейсах, входных данных, трассах выполнения, сбоях) и последующей автоматизированной разработки и непрерывной модификации вредоносных программ, используемых при реализации компьютерных атак. Противодействуют этому процессы динамического анализа, позволяющие на ранних этапах разработки выявить ошибки и уязвимости, которые могут быть использованы для реализации угроз безопасности информации на этапе эксплуатации программного обеспечения.

Технологии динамического анализа могут быть применены на разных уровнях тестирования с использованием разных подходов при реализации процессов разработки безопасного программного обеспечения, предусмотренных требованиями ГОСТ Р 56939-2024 и предполагающих тестирование программного обеспечения в динамике.

Настоящий стандарт входит в комплекс стандартов, направленных на достижение целей, связанных с предотвращением появления и устранением уязвимостей программ, и применяется совместно с ГОСТ Р 56939-2024.

Совокупность требований ГОСТ Р 56939-2024, а также настоящего стандарта и других национальных стандартов, раскрывающих требования к отдельным процессам разработки безопасного программного обеспечения, является основой превентивных

мер, направленных на противодействие угрозам безопасности информации, способы реализации которых совершенствуются в связи с развитием технологий искусственного интеллекта.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Технологии динамического анализа программного обеспечения.

Общие требования

Information protection. Secure software development.
Dynamic software analysis technologies. General requirements

Дата введения - _____

1 Область применения

Настоящий стандарт устанавливает общие требования к технологиям динамического анализа, внедрению и регулярному выполнению процессов разработки безопасного программного обеспечения, в которых применяются технологии динамического анализа, инструментальным средствам и специалистам, задействованным в указанных процессах.

Технологии динамического анализа могут применяться при реализации таких процессов, предусмотренных ГОСТ Р 56939-2024, как динамический анализ, включая фаззинг-тестирование, функциональное тестирование, нефункциональное тестирование.

Применение настоящего стандарта позволит обеспечить соответствие программного обеспечения предъявляемым к нему требованиям, в том числе требованиям по безопасности, повысить надежность программного обеспечения, снизить риски реализации угроз безопасности и некорректного функционирования программного обеспечения и, как следствие, повысить его защищенность в целом.

Настоящий стандарт предназначен:

- для организаций-разработчиков безопасного программного обеспечения;
- для организаций-разработчиков инструментальных средств, реализующих технологии динамического анализа;
- для организаций, выполняющих оценку соответствия процессов разработки безопасного программного обеспечения.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ 19.101-2024 Единая система программной документации. Виды программ и программных документов

ГОСТ 19781-90 «Обеспечение систем обработки информации программное. Термины и определения»

ГОСТ 19.102-77 Единая система программной документации. Стадии разработки

ГОСТ Р 56939-2024 «Защита информации. Разработка безопасного программного обеспечения. Общие требования»

ГОСТ Р 56920-2024 «Системная и программная инженерия. Тестирование программного обеспечения. Общие положения»

ГОСТ Р 56921-2016/ISO/IEC/IEEE 29119-2:2013 «Системная и программная инженерия. Тестирование программного обеспечения. Часть 2. Процессы тестирования»

ГОСТ Р 56922-2016/ISO/IEC/IEEE 29119-3:2013 «Системная и программная инженерия. Тестирование программного обеспечения. Часть 3. Документация тестирования»

ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»

ГОСТ Р ИСО/МЭК 27000-2021 Информационные технологии. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Общий обзор и терминология

ГОСТ Р 71206-2024 «Защита информации. Разработка безопасного программного обеспечения. Безопасный компилятор языков C/C++. Общие требования»

ГОСТ Р 58143-2018 «Информационная технология. Методы и средства обеспечения безопасности. Детализация анализа уязвимостей программного обеспечения в соответствии с ГОСТ Р ИСО/МЭК 15408 и ГОСТ Р ИСО/МЭК 18045. Часть 2. Тестирование проникновения»

ГОСТ Р ИСО/МЭК 29119-4-2021 «Информационные технологии. Методы и средства обеспечения качества. Системы и программное обеспечение. Процессы тестирования. Часть 4. Проектирование тестов»

ГОСТ Р 59453.1-2021 «Защита информации. Формальная модель управления доступом. Часть 1. Общие положения»

ГОСТ Р 59453.3-2025 «Защита информации. Формальная модель управления доступом. Часть 3. Рекомендации по разработке»

ГОСТ Р 59453.4-2025 «Защита информации. Формальная модель управления доступом. Часть 4. Рекомендации по верификации средства защиты информации, реализующего политики управления доступом, на основе формализованных описаний модели управления доступом»

Примечание – При пользовании настоящим стандартом рекомендуется проверить действие ссылочных стандартов в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

3 Термины, определения и сокращения

3.1 В настоящем стандарте применены следующие термины с соответствующими определениями:

3.1.1 **автоматизированное тестирование:** Способ тестирования (анализа), реализуемый с использованием инструментальных средств для запуска тестирования, позволяющий минимизировать участие человека в процессе тестирования.

П р и м е ч а н и е - Тестовые примеры частично или полностью выполняются специальными инструментальными средствами, однако разработку тестовых примеров и сценариев тестирования, подготовку данных, анализ результатов выполнения может выполнять человек.

3.1.2 **автоматическое тестирование:** Способ тестирования (анализа), реализуемый с использованием инструментальных средств при наступлении определенных событий (условий), заданных человеком, но не требующий непосредственного участия человека в процессе тестирования.

П р и м е ч а н и е - Примером является автоматический запуск анализа при обновлении программного кода путем интеграции инструментальных средств с системой непрерывной интеграции.

3.1.3

датчик срабатывания ошибок: Конструкция исходного или бинарного кода, которая вставляется в программу при ее сборке из исходного кода, интерпретации или двоичной трансляции и которая при обнаружении определенной ошибки в ходе выполнения программы вызывает ее аварийное завершение и (или) выдачу диагностической информации.

[Адаптировано из ГОСТ Р 71206-2024, пункт 3.9]

3.1.4

динамический анализ: Вид работ по исследованию программы (объекта анализа), основанный на анализе поведения и кода программы в режиме непосредственного исполнения (функционирования).

[Адаптировано из ГОСТ Р 56939-2024, пункт 3.2]

П р и м е ч а н и я

1 Исполнение (функционирование) программы (объекта анализа) может осуществляться в штатном (согласно эксплуатационной документации) режиме или специализированном окружении, в том числе на некоторых наборах входных данных. В рамках динамического анализа могут быть выявлены ошибки в поведении программного обеспечения в целом, отдельных программ, программных компонентов, модулей, фрагментов программ.

2 Упомянутый в тексте настоящего стандарта динамический анализ не тождественен процессу динамического анализа, предусмотренному ГОСТ Р 56939-2024, за исключением случаев, когда по тексту стандарта в явном виде указано это соответствие. Технологии динамического анализа могут применяться как в рамках процесса динамического анализа, так и любого другого процесса, где уместно исследовать поведение и код разрабатываемой программы в динамике.

3.1.5 инструмент динамического анализа: Инструментальное средство, которое выполняет динамический анализ автоматическим или автоматизированным способом.

3.1.6

исполняемый файл: Бинарный файл, содержащий исполняемый модуль программы, который может быть запущен в среде выполнения программы.

[ГОСТ Р 71206-2024, пункт 3.17]

3.1.7

компилятор: Инструментальное средство, выполняющее компиляцию.

[ГОСТ Р 71206-2024, пункт 3.4]

3.1.8

компиляция (сборка программы): Трансляция программы из ее исходного кода на языке программирования высокого уровня в бинарный код машинного языка.

[ГОСТ Р 71206-2024, пункт 3.5]

3.1.9 критерий полноты тестового покрытия: Значение тестового покрытия, достижение которого достаточно для завершения динамического анализа объекта анализа.

3.1.10 наблюдаемое поведение: Поведение программы или иного объекта анализа, связанное с взаимодействием с внешними по отношению к объекту процессами, другими объектами, устройствами, включая ввод и вывод данных, работу с внешними областями памяти, обработку или генерацию событий или прерываний.

П р и м е ч а н и е - Для одного и того же объекта анализа наблюдаемое поведение может меняться в зависимости от того, какие средства используются для контроля за поведением программы. В частности, средства контроля за состоянием внутренних переменных или выполнения отдельных инструкций внутри программы могут сделать выполнение таких инструкций или чтение (запись) некоторых внутренних областей памяти доступным для наблюдения извне программы.

3.1.11 объект анализа: Программное обеспечение, программа, программные компоненты или программные модули, отдельные функции (фрагменты программы), подлежащих динамическому анализу.

П р и м е ч а н и е - Объект анализа является эквивалентным термином по отношению к терминам «объект тестирования» и «элемент тестирования», используемых в других стандартах, принятых для применения в Российской Федерации.

3.1.12

ожидаемый результат: Предсказанное характерное поведение программы или иного объекта анализа при определенных тестовых условиях.

[Адаптировано из ГОСТ Р 56921-2016, пункт 4.8]

П р и м е ч а н и е — Характерное поведение объекта анализа может быть описано не полностью, вместо этого может быть описан ожидаемый результат. Например, вместо точного значения результата математической функции может быть задан диапазон допустимых значений.

3.1.13 ошибка в поведении: Несоответствие наблюдаемого поведения программы или иного объекта анализа требованиям или ожиданиям пользователей, которое может являться причиной невозможности выполнения требуемых функций или нарушения информационной безопасности.

П р и м е ч а н и е — Ошибка в поведении может проявляться как отказ, некорректный результат вычислений, чрезмерное снижение производительности или увеличение времени ответа, недопустимый расход ресурсов процессора или памяти, снижение пропускной способности каналов передачи данных и иным способом.

3.1.14

ошибка в программе: Конструкция или набор конструкций в исходном или бинарном коде программы, наличие которой указывает на возможность реализации угроз безопасности информации и (или) на ошибку реализованного в программе алгоритма.

[Адаптировано ГОСТ Р 71207-2024, пункт 3.1.20]

П р и м е ч а н и е - Ошибка в программе может быть уязвимостью программы, программной закладкой, ошибкой реализованного алгоритма и приводить к утечке конфиденциальных данных, отказу программного обеспечения и пр.

3.1.15 поведение программы: Совокупность действий, выполняемых программой (или иным объектом анализа), включающая в себя получение входных данных и внешних воздействий, чтение и запись данных в ходе их обработки, взаимодействие с внешними устройствами и программами, вывод полученных результатов.

П р и м е ч а н и е - Поведение программы зависит как собственно от программы, алгоритма и описаний структур данных, так и от компиляторов и других средств программно-аппаратной платформы, включая операционную систему, на которой программа выполняется, а также от действий внешних по отношению к программе, например, от ввода данных от пользователя, обмена данными по телекоммуникационным протоколам, переключения процессов и потоков программы операционной системой.

3.1.16

поверхность атаки: Множество подпрограмм (функций) программы, обрабатывающих данные, поступающие посредством интерфейсов, непосредственно или косвенно подверженных риску атаки.

[Адаптировано ГОСТ Р 56939-2024, пункт 3.9]

3.1.17

повторное тестирование: Подтверждающее тестирование объекта анализа, проводимое для проверки успешности устранения ошибки внесенными изменениями.

[Адаптировано ГОСТ Р 56939-2024, пункт 3.13]

П р и м е ч а н и е — Повторное тестирование, как правило, выполняется повтором тестового примера (тест-кейса), который привел к ошибке. При этом для более эффективного повторного тестирования могут быть определены и проанализированы новые тестовые условия, а также разработаны новые тестовые примеры.

3.1.18

программа: Данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

[ГОСТ 19781-90, статья 1]

3.1.19

программное обеспечение: Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

[ГОСТ 19781-90, статья 2]

3.1.20

программный модуль: Программа или функционально завершённый фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

[ГОСТ 19781-90, статья 15]

3.1.21

программный компонент: Программа или совокупность программ, рассматриваемая как единое целое, выполняющая законченную функцию.

[Адаптировано из ГОСТ 19.101-2024, пункт 3.3]

П р и м е ч а н и е - Термины «компонент» и «модуль» используются как синонимы согласно некоторым стандартам в области тестирования программ, при этом в настоящем стандарте принято, что данные термины не являются синонимами: программный компонент может состоять из нескольких программных модулей, а также из нескольких программ, образующих в том числе подсистему из состава программного обеспечения.

3.1.22

разработчик программного обеспечения: Организация, выполняющая работы на всех стадиях разработки программного обеспечения или ответственная, как минимум, за такие процессы жизненного цикла, как процессы реализации и процессы поддержки программного обеспечения.

[Адаптировано из ГОСТ Р 71207-2024, пункт 3.1.25]

П р и м е ч а н и я

1 Стадии разработки — по ГОСТ 19.102-77, процессы жизненного цикла — по ГОСТ Р ИСО/МЭК 12207-2010.

2 Согласно ГОСТ Р ИСО/МЭК 27000-2021 пункт 3.50 под «организацией» понимают, помимо прочего, индивидуальных предпринимателей, компании, корпорации, фирмы, партнерства, благотворительные организации, учреждения, любые составные части и сочетания названных объектов вне зависимости от того, зарегистрирована ли организация в качестве юридического лица, является ли она государственной или частной.

3 По тексту стандарта в том же значении применяется также понятие «организация» для улучшения восприятия текста.

3.1.23

регрессионное тестирование: Тестирование, проводимое после внесения изменений в объект анализа или в его среду функционирования, с целью определения влияния или подтверждения невливания внесенных изменений на ранее протестированные функции (свойства) объекта.

[Адаптировано из ГОСТ Р 56920-2024, пункт 3.14]

П р и м е ч а н и е - Регрессионное тестирование проводится в отношении объекта анализа, в программный код которого внесены изменения, с применением ранее разработанного и используемого при тестировании предыдущих версий объекта анализа тестовых примеров (тест-кейсов) и сценариев тестирования.

3.1.24

ручное тестирование: Способ тестирования, при котором тестирование объекта анализа выполняется людьми на основе задокументированного сценария с проверкой результатов путем сравнения фактических и ожидаемых результатов.

[Адаптировано из ГОСТ Р 56920-2024, пункт 3.15]

3.1.25

санитайзер: Компонент компилятора, встраивающий датчики срабатывания ошибок в генерируемый код во время компиляции программы.

[ГОСТ Р 71206-2024, пункт 3.10]

3.1.26

сборочная среда: Совокупность программных и аппаратных средств, служб связи, интерфейсов, форматов данных, протоколов, стандартов, обеспечивающих преобразование исходного кода программ в программные модули в соответствии с представленными метаданными и с учетом зависимостей программного модуля.

[ГОСТ Р 56939-2024, пункт 3.15]

3.1.27

сценарий тестирования: спецификация ручного или автоматизированного тестирования, содержащая сведения о тестовых примерах (тест-кейсах) и последовательности выполнения.

[Адаптировано из ГОСТ Р 56921-2016, пункт 4.61]

П р и м е ч а н и е - Сценарий тестирования может быть представлен в виде документа или с использованием средств управления тестированием, систем отслеживания ошибок, электронных таблиц, информационных ресурсов организации и в ином виде.

3.1.28

тестирование: Комплекс мероприятий, проводимых для выявления и оценки функций и свойств объекта анализа.

[Адаптировано из ГОСТ Р 56920-2024, пункт 3.19]

П р и м е ч а н и е — Как правило, под тестированием предполагается проверка выполнения требований к объекту анализа при исполнении некоторого набора заранее определенных или конструируемых в ходе самого тестирования тест-кейсов.

3.1.29 тест-кейс: Набор предварительных условий, входных данных, пошаговых инструкций и ожидаемых результатов, разработанный для наблюдения за поведением программы или иного объекта анализа для достижения целей тестирования.

П р и м е ч а н и я

1 Тест-кейс является синонимом термина «тестовый пример» (ГОСТ Р 56920-2025, пункт 3.20), а также термина «контрольный пример» (ГОСТ Р 56921-2016/ISO/IEC/IEEE 29119-2:2013, ГОСТ Р 56922-2016/ISO/IEC/IEEE 29119-2:2013).

2 Тест-кейс может быть представлен как набор инструкций в виде документа для человека в случае ручного тестирования, и в виде программы в случае автоматизированного (автоматического) тестирования.

3.1.30 тестовое покрытие: Мера полноты набора тест-кейсов, которая может выражаться тестовым покрытием программного кода, тестовым покрытием требований, тестовым покрытием формальных моделей объекта анализа или иным способом.

3.1.31 тестовое покрытие программного кода: Отношение, выраженное в процентах, между количеством участков кода программы (программного модуля) определенного вида, задействованных в ходе динамического анализа, и всем совокупным количеством достижимых участков кода этого вида.

П р и м е ч а н и е — Покрытие кода может вычисляться для инструкций, базовых блоков, ветвлений в коде, функций.

3.1.32 тестовое покрытие требований: Отношение, выраженное в процентах, между количеством требований, предъявляемых к программе (программному модулю или компоненту) и тестируемых в ходе динамического анализа, и количеством всех предъявляемых к ней требований.

3.1.33 технологии динамического анализа: Совокупность видов, уровней, методов, способов и процессов динамического анализа, инструментальных средств и правил их применения.

3.1.34 уровень тестирования: реализация процесса тестирования (динамического анализа), которая зависит от уровня грануляции объекта анализа и цели тестирования и в рамках которой используются группы тест-кейсов, объединенные общим назначением.

П р и м е ч а н и е - В контексте настоящего стандарта рассматриваются три уровня тестирования: модульное тестирование, интеграционное тестирование и системное тестирование. Указанные уровни тестирования предусмотрены ГОСТ Р 56920-2024, при этом комплексное тестирование в настоящем стандарте определено как интеграционное.

3.1.35 фаззер: Инструментальное средство динамического анализа, которое выполняет фаззинг-тестирование.

3.1.36

фактические результаты: Совокупность поведения программы или совокупность условий, связанных данных или тестовой среды, полученные в результате выполнения тест-кейса.

[Адаптировано из ГОСТ Р 56921-2016, пункт 4.1]

3.1.37 юнит-тест: Тест-кейс, который проверяет программу или отдельные программные модули в изоляции от других модулей и программ.

3.2 В настоящем стандарте применены следующие обозначения и сокращения:

- | | | |
|-----|---|---|
| СЗИ | — | средство защиты информации; |
| ПО | — | программное обеспечение; |
| API | — | Application Programming Interface (прикладной программный интерфейс), описание способов (набор классов, процедур, функций, структур или констант), с помощью которых одна программа может взаимодействовать с другими программами). |

4 Общие положения

4.1 Предотвращение появления уязвимостей ПО достигается путем реализации разработчиком ПО совокупности процессов разработки безопасного ПО, представленных в ГОСТ Р 56939-2024.

4.2 Выявление явных или скрытых недостатков, в том числе уязвимостей, и недеklarированных возможностей при функционировании ПО (исполнении кода

программ), обеспечивается путем внедрения и регулярного использования технологий динамического анализа в процессах разработки безопасного ПО.

4.3 Положения, представленные в настоящем стандарте, выражены в форме требования, рекомендации или возможного действия, предназначенных для достижения результатов применения технологий динамического анализа. Для этой цели в настоящем стандарте используют вспомогательные глаголы «должен», «рекомендуется» и «может», чтобы подчеркнуть различие между разными формами требований к реализации мер. Глагол «должен» использован для выражения требования, обязательного к реализации, «рекомендуется» – для выражения требования, рекомендованного к реализации, «может» – для того, чтобы отразить возможные направления реализации требований.

4.4 Технологии динамического анализа могут применяться на разных этапах жизненного цикла ПО при реализации таких процессов, предусмотренных ГОСТ Р 56939-2024, как динамический анализ и фаззинг-тестирование, функциональное тестирование, нефункциональное тестирование. Сведения о применимости технологий динамического анализа в указанных процессах приведены в виде схемы в Приложении А.

4.5 Технологии динамического анализа могут применяться на разных уровнях тестирования, выбор которого зависит от объекта анализа и целей тестирования.

В настоящем стандарте вводятся три уровня тестирования:

- 1) модульное тестирование;
- 2) интеграционное тестирование;
- 3) системное тестирование.

Модульное тестирование предполагает реализацию процесса динамического анализа ПО на уровне и в отношении программных модулей или отдельных фрагментов программ отдельно (изолировано) от других модулей, компонентов и программ с целью выявления ошибок в программе. При модульном тестировании используются API-интерфейсы программных модулей.

Интеграционное (комплексное) тестирование предполагает реализацию процесса динамического анализа ПО в отношении нескольких взаимосвязанных составных частей ПО (подсистем, программных компонентов, программ, программных модулей) с целью определения корректности их взаимодействия и выявления ошибок в поведении.

Системное тестирование предполагает реализацию процесса динамического анализа на уровне и в отношении всего ПО в целом с целью определения соответствия заданным требованиям и выявления ошибок в поведении. При системном тестировании, как правило, используются внешние API и пользовательские интерфейсы ПО. В отличие от интеграционного тестирования при системном тестировании проверки проводятся не в отношении взаимосвязанных составных частей ПО, а в отношении взаимодействия и интеграции ПО с аппаратными средствами, внешним ПО, информационными системами, пользователями.

4.6 Настоящий стандарт не устанавливает требования о применении всей совокупности представленных технологий динамического анализа.

Разработчик ПО должен осуществлять выбор технологий динамического анализа, включая уровни, методы и способы тестирования, для использования в процессах разработки безопасного ПО на основании установленных им критериев.

Критерии могут основываться на требованиях нормативных документов и других национальных стандартов в области защиты информации и разработки безопасного ПО, учитывать возможность и целесообразность применения технологий в отношении разрабатываемого ПО.

При выборе технологий динамического анализа также рекомендуется учитывать лучшие практики в области разработки безопасного ПО. В частности, функциональное тестирование на модульном уровне и системном уровнях, фаззинг-тестирование являются базовыми технологиями, составляющими основы тестирования ПО в динамике.

Технологии и методы динамического анализа, представленные в настоящем стандарте, не являются исчерпывающими.

4.7 Разработчик ПО должен выполнять динамический анализ ПО, анализ результатов и устранение выявленных недостатков в соответствии с организационно-распорядительной документацией.

4.8 Общие требования к организации динамического анализа приведены в разделе 5.

4.9 Внедрение и применение технологий динамического анализа и использующих их процессов разработки ПО в общем случае характеризуется последовательностью этапов и действий, определенных в разделах 6 и 7.

Данные этапы и действия могут иметь свою специфику в зависимости от технологии динамического анализа, которая учитывается в требованиях раздела 8.

Диаграммы процессов внедрения технологий и регулярного выполнения динамического анализа в общем виде представлены в Приложениях Б и В.

4.10 Описание и требования к технологиям динамического анализа с учетом особенностей их внедрения и регулярного выполнения представлены в разделе 8. Данные требования дополняют общие требования, содержащиеся в разделах 6 и 7.

4.11 Требования к инструментам динамического анализа представлены в разделе 9.

4.12 Требования к специалистам, участвующим в процессах динамического анализа, представлены в разделе 10.

5 Общие требования к организации динамического анализа

5.1 На организационном этапе разработчик ПО должен определить и задокументировать общие требования к тестированию ПО на уровне организации, не специфичные для конкретного ПО и применимые для всего ПО из области действия процессов разработки безопасного ПО.

5.2 Общие требования должны содержать, как минимум, следующие сведения:

- 1) цели тестирования;
- 2) область применения процессов динамического анализа;
- 3) общие сведения и перечень процессов и технологий динамического анализа, применяемых в организации;
- 4) критерии выбора технологий динамического анализа и уровня тестирования, способов тестирования, которыми необходимо руководствоваться при планировании динамического анализа ПО организации;
- 5) критерии выбора объектов анализа для каждой применяемой технологии динамического анализа с учетом уровня тестирования.

Общие требования должны также содержать в отношении каждой применяемой технологии динамического анализа с учетом уровня тестирования следующие сведения:

- 1) этапы жизненного цикла ПО, на которых выполняется тестирование;
- 2) требования к автоматизации;
- 3) требуемая степень участия или независимости участников при реализации технологий.

П р и м е ч а н и е - Требуемая степень участия или независимости участников анализа предполагает указания в явном виде, какие технологии и на каких уровнях должны быть реализованы в процессе разработки непосредственно разработчиками программного кода, а другие — тестировщиками или специалистами по информационной безопасности. То же — в части верификации результатов тестирования (если применимо).

5.3 Общие требования могут быть представлены в руководстве по разработке безопасного ПО, в регламентах отдельных процессов разработки безопасного ПО или в виде самостоятельного документа.

П р и м е ч а н и е - Приведенные требования сопоставимы с требованиями к организационной спецификации тестирования и политике тестирования согласно ГОСТ Р 56922-2016/ISO/IEC/IEEE 29119-3:2013.

6 Общие требования к внедрению технологий динамического анализа

6.1 Внедрение каждой технологии динамического анализа условно состоит из следующих этапов:

- 1) начальный этап внедрения;
- 2) подготовительный этап внедрения;
- 3) выполнение первичного динамического анализа;
- 4) организация регулярного динамического анализа.

Указанные этапы в общем случае являются применимыми для всех технологий динамического анализа, при этом реализация некоторых этапов имеет свои особенности, которые зависят от требований к определенной технологии и приведены в разделе 8.

Диаграмма процесса внедрения динамического анализа в общем виде представлена в Приложении Б.

6.2 Требования к начальному этапу внедрения.

6.2.1 Разработчик ПО должен определить ПО и технологию динамического анализа (включая уровень тестирования) для внедрения.

П р и м е ч а н и е - Выбор ПО и технологии динамического анализа на этапе внедрения зависят от планов развития (реализации) процессов разработки безопасного ПО разработчика ПО, в соответствии с которыми разные технологии могут быть внедрены как последовательно, так и параллельно, в отношении одного или нескольких ПО.

6.2.2 Разработчик ПО должен выполнить сбор и анализ имеющихся сведений о ПО с целью получения представления о назначении и архитектуре ПО, системных требованиях и среде (средах) функционирования, функциональных и нефункциональных требованиях и возможностях ПО (включая требования безопасности), сценариях и условиях применения ПО, используемых протоколах и интерфейсах взаимодействия ПО с пользователями и внешним ПО, о языках программирования, с использованием которых написан программный код ПО, о поверхности атаки ПО.

П р и м е ч а н и я

1 Источниками сведений могут являться описание архитектуры ПО и поверхности атаки, набор требований безопасности ПО, модель угроз и перечень мер по нейтрализации угроз безопасности, перечень программных компонентов (перечень зависимостей), сформированные в качестве артефактов соответствующих процессов согласно ГОСТ Р 56939-2024, проектная, программная, эксплуатационная или другая документация ПО, другие источники информации о ПО.

6.2.3 Разработчик ПО должен на основании анализа сведений о ПО сформировать спецификацию функциональных и нефункциональных требований к ПО, включая требования безопасности ПО.

В спецификации рекомендуется выделять требования высокого и низкого уровней:

1) требования высокого уровня определяют правила реализации, использования или взаимодействия с ПО (в целом и с его отдельными подсистемами);

2) требования низкого уровня детализируют требования высокого уровня и могут включать правила использования и взаимодействия для отдельных программ, программных модулей, функций, а также отдельных процедур, методов, включая детали реализации (например, типов данных или алгоритмов).

Зависимости между требованиями высокого и низкого уровня должны быть специфицированы.

Каждое требование должно быть сформулировано и иметь свой уникальный идентификатор.

Требования должны быть сформулированы таким образом, чтобы обеспечивались их однозначность, полнота, непротиворечивость другим требованиям и проверяемость.

Идентификатор требования должен быть сформирован таким образом, чтобы обеспечить сопоставимость между требованиями высокого и низкого уровней. Идентификатор требования должен использоваться разработчиком ПО при

реализации требования (разработке программного кода) и тестирования реализации требования.

П р и м е ч а н и е - Например, идентификатор требования может быть указан в задачах на разработку и тестирование кода.

Систематизацию, хранение и отслеживание изменений требований рекомендуется выполнять с помощью специализированных инструментальных средств управления требованиями.

П р и м е ч а н и е - В случае, если ПО реализует функции управления доступом, в качестве метода систематизации требований безопасности, связанных реализацией ПО таких функций, могут использоваться формальные модели, разрабатываемые в соответствии с ГОСТ Р 59453.1-2021, ГОСТ Р 59453.3-2025.

6.3 Требования к подготовительному этапу внедрения.

6.3.1 Разработчик ПО должен определить в соответствии с установленными в общих требованиях к тестированию ПО критериями объект (объекты) анализа и соответствующий уровень тестирования.

6.3.2 Разработчик ПО должен выполнить сбор дополнительных сведений об объекте анализа, если это предусмотрено требованиями к технологии динамического анализа в разделе 8.

6.3.3 Если требованиями к технологии динамического анализа определена необходимость выполнения тестовой сборки объекта анализа с встроенными датчиками срабатывания ошибок, разработчик ПО должен:

- выполнить сбор и анализ сведений о системе сборки и сборочной среде программных модулей, системе и инструментах сборки (в т.ч. компиляторах);
- определить особенности, которые необходимо учесть при формировании сборочного окружения и сборке;
- определить возможности и способы встраивания датчиков срабатывания ошибок, возможности использования санитайзеров, определить необходимые флаги компиляции.

П р и м е ч а н и я

1 Возможность встраивания датчиков срабатывания ошибок зависит от языков программирования, применяемых компиляторов и среды функционирования.

2 Перечень требуемых датчиков ошибок (ошибки работы с памятью, неопределенное поведение программы, нарушения потока управления и т. п.) определяются исходя из технических возможностей, обусловленных используемыми языками программирования и средой функционирования.

Если система сборки для среды функционирования не позволяет штатным образом встраивать датчики срабатывания ошибок, разработчик ПО должен определить возможность использования других средств и методов

П р и м е ч а н и е - Например, динамического двоичного инструментария и динамической компоновки, отладочных версий менеджеров памяти, подключаемых на этапе компоновки или в рамках симулятора, запуска интерпретатора с дополнительными опциями.

Если для языка программирования ПО или среды функционирования датчики срабатывания ошибок не применимы или отсутствуют, указанное ограничение должно быть зафиксировано и компенсировано исследованиями в рамках процессов разработки безопасного ПО.

6.3.4 Разработчик ПО должен определить и задокументировать в организационно-распорядительной документации критерии полноты и способы сбора тестового покрытия. Определение критериев полноты тестового покрытия должно осуществляться с учетом уровня тестирования и требований к технологии динамического анализа, установленных в разделе 8.

При этом на модульном уровне тестирования при применении любой технологии разработчик ПО должен выполнять, как минимум, сбор тестового покрытия программного кода. Сбор покрытия программного кода может осуществляться для исходного или бинарного кода объекта анализа. Случаи, при которых проводится сбор покрытия только по бинарному коду, должны быть обоснованы разработчиком ПО. Минимально требуемое покрытие программного кода должно измерять покрытие инструкций или базовых блоков, более детальное — может измерять покрытие ветвлений или последовательностей исполняемых базовых блоков, покрытие комбинаций элементарных логических условий в составных логических выражениях в условиях ветвлений.

6.3.5 Разработчик ПО должен на основании анализа полученных сведений о ПО сформировать требования, предъявляемые к тестовой среде (для каждого ПО и внедряемой технологии динамического анализа с учетом уровня тестирования).

П р и м е ч а н и е - Требования к тестовой среде в общем случае могут предъявляться к вычислительным ресурсам, сетевой изоляции или интеграции с другими системами посредством определенных интерфейсов взаимодействия, среде функционирования, хранилищу тестовых данных и результатов тестирования, отказоустойчивости, постоянной или временной доступности и иные.

Тестовая среда должна быть изолирована от среды, в которой осуществляется непосредственно разработка программного кода, и от среды эксплуатации ПО (если применимо).

При формировании требований разработчик ПО должен учитывать требования к внедряемой технологии, установленные в разделе 8.

6.3.6 Разработчик ПО должен сформировать и задокументировать в организационно-распорядительной документации критерии выбора инструментов динамического анализа на основании требований к инструментам динамического анализа, установленных в разделе 9, и с учетом их применимости для реализации выбранной технологии динамического анализа (включая уровни, методы и способы тестирования) в отношении ПО из области применения процессов динамического анализа.

Так как в процессах динамического анализа могут применяться инструменты разного назначения (например, средства сбора тестового покрытия, средства генерации тестовых данных и тест-кейсов, средства управления и автоматизации тестирования, фаззеры и иные), критерии выбора должны содержать как общие для всех видов инструментов динамического анализа, так и частные, применимые в отношении инструментов определенного назначения.

6.3.7 Разработчик ПО должен осуществить выбор инструментов динамического анализа для каждого ПО и внедряемой технологии динамического анализа:

- а) в соответствии с установленными критериями выбора;
- б) с учетом сведений о ПО, полученных по результатам сбора и анализа сведений о ПО, а именно:
 - сведений о языках программирования, используемых при разработке ПО;
 - сведений о системе сборки, инструментальных средствах сборки (в частности, компиляторах);
 - среде функционирования ПО;
- в) с учетом установленных требований к сбору тестового покрытия.

Функциональные возможности инструментов динамического анализа должны обеспечивать возможность реализации внедряемой технологии динамического анализа в отношении ПО с учетом предъявляемых к ней в разделе 8 требований.

П р и м е ч а н и е - Примером средств сбора тестового покрытия по исходному коду являются утилиты gcov, lcov, llvm-cov, по бинарному коду — утилита drcov.

При выборе фаззера разработчик ПО должен учитывать результаты сравнительной оценки фаззеров в части динамики роста тестового покрытия программного кода.

П р и м е ч а н и е - Такие результаты могут быть получены на основании анализа сведений, доступных в открытом доступе, или от производителей фаззеров, а также самостоятельно разработчиком ПО путем использования сервисов для сравнительной оценки фаззеров. Примером публичного сервиса сравнительной оценки фаззеров является FuzzBench.

При выборе проприетарных инструментов динамического анализа рекомендуется отдавать предпочтение средствам российских разработчиков.

6.3.8 Разработчик ПО должен определить и задокументировать в организационно-распорядительной документации критерии пересмотра набора используемых инструментов динамического анализа и их конфигураций.

П р и м е ч а н и е - В качестве критериев пересмотра набора используемых инструментов динамического анализа и (или) их конфигураций могут использоваться следующие:

- изменение архитектуры ПО и (или) поверхности атаки, в результате которых изменяются состав программных модулей ПО и состав исследуемых интерфейсов, в состав ПО включаются программные модули, написанные с использованием нового языка программирования;
- изменение среды, условий и сценариев функционирования ПО;
- изменение тестовой среды и (или) сборочной среды, системы сборки (если применимо);
- обновление версии инструментов динамического анализа;
- изменение нормативных правовых актов и стандартов в области динамического анализа.

6.3.9 Разработчик ПО должен определить и задокументировать в организационно-распорядительной документации требования к тест-кейсам и тестовым данным (наборам данных), сценариям тестирования с учетом требований к технологиям динамического анализа, указанным в разделе 8.

Общими обязательными требованиями к сценариям тестирования и тест-кейсам являются:

- наличие идентификатора каждого сценария тестирования и каждого тест-кейса;
- прослеживаемость между каждым сценарием тестирования и каждым тест-кейсом к тестируемым требованиям высокого или низкого уровней с использованием идентификатора требования (при тестировании на основе требований) или описание назначения каждого сценария тестирования и каждого тест-кейса, определенное с учетом целей применения других технологий динамического анализа;
- наличие предварительных условий;
- последовательность выполнения действий в рамках тест-кейса или сценария тестирования;
- ожидаемые результаты и (или) критерии завершения.

Требованиями к сценариям тестирования (тест-кейсам) должны учитывать требования, установленные ГОСТ Р 56939-2024 к сценариям тестирования в рамках динамического анализа (пункт 5.11.3.4).

6.3.10 Разработчик ПО должен разработать тест-кейсы и сценарии тестирования, подготовить тестовые данные в соответствии с установленными требованиями.

Рекомендуется в первую очередь разрабатывать юнит-тесты и функциональные тест-кейсы, так как в процессе их разработки могут быть выявлены недостатки архитектуры или реализации функциональных требований ПО, а также данные тест-кейсы могут использоваться как задел при внедрении других технологий динамического анализа.

При разработке тест-кейсов рекомендуется учитывать необходимость их полной или частичной автоматизации в целях регулярного динамического анализа, в частности при проведении регрессионного тестирования.

6.3.11 Разработчик ПО должен обеспечить хранение и версионирование подготовленных сценариев тестирования, тест-кейсов и наборов тестовых данных.

6.3.12 Разработчик ПО должен определить и задокументировать в организационно-распорядительной документации критерии пересмотра и актуализации сценариев тестирования, тест-кейсов и тестовых данных.

П р и м е ч а н и е - В качестве критериев могут использоваться следующие события: изменение назначения, функциональных возможностей ПО, состава и формулировок требований к ПО, архитектуры ПО и (или) поверхности атаки, изменение среды или условий функционирования ПО.

6.3.13 Разработчик ПО должен подготовить тестовую среду в соответствии с установленными к ней требованиями.

Подготовка тестовой среды в общем случае предполагает создание всех необходимых условий для выполнения динамического анализа: реализацию требований к тестовой среде, установку и настройку необходимого системного и иного ПО, инструментов динамического анализа.

В ходе настройки инструментов динамического анализа разработчик ПО должен выполнить их конфигурирование таким образом, чтобы обеспечить регистрацию информации о ходе и результатах тестирования (в журналах тестирования или другим способом), тестовом покрытии.

Сведения о конфигурации инструментов или их конфигурационные файлы должны быть сохранены.

6.3.14 Разработчик ПО должен подготовить среду сборки и сборочное окружение, позволяющее выполнить тестовую сборку объекта анализа, если это предусмотрено требованиями к технологии динамического анализа, установленными в разделе 8.

6.3.15 По результатам подготовки тестовой среды разработчик ПО должен сформировать:

1) схему и (или) описание тестовой среды;

2) перечень инструментов динамического анализа, содержащий сведения согласно ГОСТ Р 56939-2024 (пункт 5.11.3.2), включая сведения о назначении инструментов динамического анализа, их сопоставлении с объектами анализа и языками программирования, о конфигурациях (опциях) запуска и особенностях эксплуатации.

Рекомендуется формировать описание тестовой среды таким образом, чтобы оно было достаточным для воспроизведения тестовой среды или восстановления ее в исходное состояние по завершении тестирования.

6.4 Требования к первичному динамическому анализу.

6.4.1 Разработчик ПО должен определить для каждой технологии и задокументировать в организационно-распорядительной документации:

- критерии или стартовые события начала динамического анализа;
- критерии завершения динамического анализа и (или) условия, при которых его результаты признаются успешными;
- требования к результатам динамического анализа;
- требования к хранению результатов динамического анализа.

П р и м е ч а н и е - При формировании стартовых событий начала динамического анализа рекомендуется учитывать, что модульное и интеграционное тестирование может проводиться при готовности объектов анализа даже в случае, если разработка ПО в целом не завершена.

6.4.2 Разработчик ПО должен выполнить тестовую сборку объекта анализа, если это предусмотрено требованиями к технологии динамического анализа, установленными в разделе 8.

6.4.3 Разработчик ПО должен выполнить динамический анализ объекта анализа в подготовленной тестовой среде (для каждого ПО и внедряемой технологии динамического анализа) путем запуска инструментов динамического анализа и разработанных тест-кейсов. Особенности выполнения динамического анализа в зависимости от выбранной технологии приведены в разделе 8.

6.4.4 Разработчик ПО должен обеспечить сбор тестового покрытия с учетом требований к технологии, установленных в разделе 8.

6.4.5 Разработчик ПО должен сохранить результаты первичного динамического анализа.

6.4.6 Разработчик ПО должен выполнить анализ результатов динамического анализа и сформировать выводы о качестве выполненного динамического анализа (для каждого ПО и внедряемой технологии динамического анализа) на основании следующих критериев:

- инструменты динамического анализа функционируют корректно, в том числе с учетом временных и иных ограничений (при наличии);
- сбор тестового покрытия выполняется, требуемое тестовое покрытие достигается или (для определенных технологий) выполняются критерии завершения тестирования;
- ход выполнения анализа регистрируется, формируются отчеты и журналы тестирования, содержащие необходимые сведения.

В случае, если хотя бы один из критериев не достигается, разработчик ПО должен принять меры, в том числе по уточнению выбора инструментов динамического анализа, пересмотру конфигурации тестовой среды и инструментов динамического анализа, используемых тест-кейсов и тестовых данных, далее выполнять динамический анализ до тех пор, пока все три критерия не будут достигнуты.

6.5 Требования к организации регулярного динамического анализа.

6.5.1 Разработчик ПО должен определить в отношении каждой внедряемой технологии динамического анализа и задокументировать в организационно-распорядительной документации:

1) критерии, в соответствии с которыми определяется приоритет при выполнении динамического анализа, обработке и анализе его результатов, устранении выявленных недостатков;

П р и м е ч а н и е - Например, критерии могут быть определены на основании тестируемых требований, функциональных возможностей, классификации программных модулей и их программных интерфейсов и на иных основаниях.

3) порядок классификации недостатков и определения критичности ошибок, выявляемых в рамках динамического анализа;

3) периодичность динамического анализа, включая критерии повторного и регрессионного тестирования и события, при наступлении которых требуется выполнение динамического анализа;

4) требования к отчетности.

При определении критериев повторного и регрессионного тестирования разработчик ПО должен учитывать необходимость выполнения повторного тестирования объекта анализа после устранения недостатков, а также регрессионного тестирования при внесении изменений в программный код (при доработке) объекта анализа.

К событиям, при наступлении которых необходимо выполнение динамического анализа, должны относиться, как минимум, следующие:

1) изменение архитектуры ПО в целом или отдельного объекта анализа;

2) изменение состава функциональных и нефункциональных требований и возможностей ПО или способа реализации этих требований;

3) изменение среды и условий функционирования ПО в целом или отдельного объекта анализа;

4) обновление или изменение конфигураций используемых инструментов динамического анализа.

Требования к отчетности должны учитывать требования к отчетам, как артефактам процессов ГОСТ Р 56939-2024, в рамках которых применяются технологии динамического анализа. Требования к отчетности должны включать указания о включении в отчет следующих сведений:

- наименование, идентификатор (или версию) ПО;
- наименование и идентификатор (или версию) объекта (объектов) анализа (для объектов анализа, отличных от ПО в целом);
- сведения об используемых инструментах динамического анализа;
- описание среды функционирования;
- перечень выполненных тест-кейсов (сценариев тестирования) и последовательность их выполнения;
- описание ограничений или отдельных аспектов, которые не проверялись;
- описание полученных результатов, перечень обнаруженных ошибок и результатов их анализа;
- выводы.

Данный перечень может быть дополнен или скорректирован с учетом

технологии динамического анализа и уровня тестирования (например, отчет фаззинг-тестирования должен содержать сведения о данных, подача которых привела к ошибкам в поведении программы или аварийному завершению).

6.5.2 Разработчику ПО рекомендуется определить и задокументировать в организационно-распорядительной документации метрики и целевые показатели качества динамического анализа как процесса.

П р и м е ч а н и е — Показатели и метрики качества динамического анализа как процесса могут быть определены на основании достигаемого тестового покрытия и количества выполненных тест-кейсов, количества выявленных и устраненных ошибок, количества повторно выявленных ошибок, времени, потраченного на выполнение анализа и устранение ошибок.

6.5.3 Разработчик ПО должен определить автоматизируемые функции процессов динамического анализа с учетом особенностей технологии динамического анализа и спланировать соответствующие работы по автоматизации для обеспечения регулярного выполнения динамического анализа.

П р и м е ч а н и е - Автоматизируемыми функциями могут являться разработка юнит-тестов, тест-кейсов, подготовка тестовых (входных) данных, запуск инструментальных средств, передача отчетов инструментальных средств в систему отслеживания ошибок и иные.

6.5.4 Разработчик ПО должен разработать или уточнить организационно-распорядительную документацию по результатам внедрения технологий динамического анализа.

Организационно-распорядительная документация процессов динамического анализа:

- 1) должна содержать сведения согласно требованиям ГОСТ Р 56939-2024, предъявляемым к регламентам соответствующих процессов как артефактов;
- 2) должна содержать сведения, определенные и задокументированные разработчиком ПО на этапе внедрения динамического анализа;
- 3) должна определять порядок действий при выполнении регулярного динамического анализа (с учетом требований разделов 7 и 8);
- 4) должна быть согласована ответственными лицами и доведена до всех участников процесса;
- 5) может быть представлена в виде набора регламентов, инструкций, методик, моделей и описания моделей процессов на бумажных носителях или в электронном виде, включая размещение на информационных ресурсах разработчика ПО.

7 Общие требования к регулярному выполнению динамического анализа

7.1 Разработчик ПО должен выполнять регулярный динамический анализ в соответствии с организационно-распорядительной документацией.

7.2 Разработчик ПО должен поддерживать в актуальном состоянии организационно-распорядительную документацию процессов динамического анализа, обеспечить версионирование и контроль изменений организационно-распорядительной документации,

7.3 Перед выполнением регулярного динамического анализа разработчику ПО рекомендуется формировать план тестирования, содержащий следующие сведения:

- 1) наименование ПО;
- 2) цель тестирования;
- 3) технология, уровень, методы и способы тестирования;
- 4) сведения об организационно-распорядительной документации, которой необходимо руководствоваться в процессе выполнения динамического анализа;
- 5) критерии начала тестирования;
- 6) критерии полноты тестового покрытия и (или) завершения тестирования;
- 7) объем тестирования (состав тестируемых требований или объектов анализа, или другие сведения);
- 8) описание тестовой среды;
- 9) ожидаемые результаты тестирования;
- 10) сведения об используемых тест-кейсах и (или) сценариях тестирования;
- 11) технические и временные ограничения (если применимо).
- 12) исполнители и сроки.

План может содержать иные сведения, определяемые разработчиком ПО как необходимые.

Рекомендуется формировать отдельный план для каждой технологии. План может быть представлен в виде документа, в электронном виде на корпоративных информационных ресурсах организации-разработчика, в виде проекта в системе управления проектами (задачами), в виде проекта в системе управления тестированием и ином виде.

Необходимость разработки плана тестирования рекомендуется определять в зависимости от:

- объекта анализа;
- применяемой технологии, уровня и способов динамического анализа;
- этапа жизненного цикла ПО, на котором выполняется динамический анализ;
- начальных событий, запускающих процесс регулярного динамического анализа;
- наличия изменений в ПО (объекте анализа);
- других критериев, определяемых разработчиком ПО.

П р и м е ч а н и е - В случае, если тестирование выполняется автоматизированным или автоматическим способом при внесении изменений в исходный код, формирование плана тестирования может быть нецелесообразно. В случае, когда динамический анализ предполагается в отношении ПО в целом при подготовке к выпуску новой версии ПО или по результатам изменений функциональных возможностей или архитектуры ПО, планирование может быть необходимым.

7.4 Перед выполнением регулярного динамического анализа разработчик ПО должен выполнить анализ изменений ПО и (или) других объектов анализа по сравнению с версией, в отношении которой ранее был проведен динамический анализ (в том числе, в рамках внедрения или предыдущей версии объекта анализа).

При необходимости разработчик ПО должен:

- 1) уточнить сведения о ПО, включая спецификацию требований;
- 2) уточнить состав объектов анализа, подлежащих тестированию;
- 3) собрать или уточнить сведения об объектах анализа;
- 4) определить достаточность и актуальность сценариев тестирования и (или) тест-кейсов для выполнения тестирования объекта анализа и достижения тестового покрытия требований, при недостаточности или неактуальности тест-кейсов — доработать их;
- 5) выполнить дополнительную настройку или создать новую тестовую среду (при необходимости).

Особенности выполнения требований в зависимости от применяемой технологии динамического анализа приведены в разделе 8.

Работы по анализу изменений, уточнению сведений и доработке тест-кейсов в таком случае рекомендуется включать в план тестирования (при его разработке) или учитывать их при определении сроков тестирования.

7.5 При выполнении регулярного динамического анализа должны обеспечиваться регистрация событий выполнения динамического анализа, сбор тестового покрытия (если оно предусмотрено технологией динамического анализа

согласно требованиям, установленным в разделе 8), сохранение результатов.

7.6 При выполнении регулярного динамического анализа разработчик ПО должен выполнять анализ результатов динамического анализа, а именно:

1) оценить полноту достигнутого тестового покрытия или достижение условий успешного завершения тестирования, определенных в организационно-распорядительной документации;

2) выполнить обработку срабатываний инструментов динамического анализа, определить тип ошибок, по результатам которой:

а) в отношении ошибок, связанных с некорректными тест-кейсами (включая синтетические цели и функции-обертки) и тестовыми данными, некорректной конфигурацией запуска инструментов, некорректной настройкой среды функционирования (выполнения) или тестовой среды, — устранить ошибки непосредственно после выявления;

б) в отношении ошибок в поведении — выполнить анализ полученных результатов, определить причины ошибки и ее критичность, предпринять действия в соответствии с принятым в организации порядком управления недостатками с учетом приоритетов, установленных в организационно-распорядительной документации.

Примечания

1 Такой порядок может быть определен в регламенте управления недостатками, предусмотренным ГОСТ Р 56939-2024 (пункт 5.3.1).

2 Для устранения ошибок в поведении, вызванных ошибками в программе, необходимо выявить ошибку в программе и исправить ее или устранить условия выполнения программы, при которых ошибка происходит, например, ограничив класс входных данных или исключив возможность использования некоторых интерфейсов программы.

7.7 В случае внесения изменений в программный код в целях устранения ошибок в программе, разработчик ПО должен отметить выполненные изменения исходного кода ПО в системе контроля версий.

7.8 Разработчику ПО рекомендуется при выполнении анализа результатов динамического анализа выявлять участки кода программы, незадействованные при выполнении динамического анализа, и проводить в отношении них дополнительные исследования. При исследованиях непокрытого кода рекомендуется учитывать, что он может как реализовывать специфичные функциональные возможности (включая «забытые» требования или детали требований, оптимизировать реализацию определенных алгоритмов), так и являться избыточным, не используемым при выполнении программы или не влияющим на результат ее выполнения. В первом

случае необходимо обеспечить покрытие такого кода с помощью дополнительных тест-кейсов, во втором - исправить или исключить код, так как его наличие может приводить к возникновению недеklarированных возможностей и уязвимостей в ПО, нецелесообразному увеличению объема кода и затруднению его поддержки.

7.9 Разработчик ПО должен выполнить повторное тестирование после устранения выявленных недостатков с использованием тест-кейсов и тестовых данных, при использовании которых они выявлены.

7.10 Разработчик ПО должен формировать отчеты по результатам динамического анализа в соответствии с установленными в организационно-распорядительной документации требованиями к отчетности.

7.11 Разработчик ПО должен учитывать результаты динамического анализа в процессе обеспечения безопасности при выпуске готовой к эксплуатации версии ПО, предусмотренном ГОСТ Р 56939-2024 (пункт 5.20), в частности в типовых сценариях приемки ПО (версии ПО) перед предоставлением пользователю.

7.12 Разработчик ПО должен поддерживать в актуальном состоянии сведения о ПО и спецификацию требований на протяжении всего жизненного цикла ПО (версии ПО) до вывода ПО (версии ПО) из эксплуатации.

7.13 Разработчик ПО должен пересматривать и дорабатывать тестовые данные, тест-кейсы, сценарии тестирования в соответствии с установленными в организационно-распорядительной документации критериями.

7.14 Разработчик ПО должен пересматривать перечень и конфигурацию инструментов динамического анализа в соответствии с установленными в организационно-распорядительной документации критериями.

7.15 Результаты, полученные в рамках регулярного динамического анализа, рекомендуется учитывать при планировании развития процессов разработки безопасного ПО, их улучшения и автоматизации.

8 Описание и требования к технологиям динамического анализа

8.1 Тестирование на основе требований: функциональное тестирование

8.1.1 Функциональное тестирование применяется для проверки соответствия объекта анализа предъявляемым к нему функциональным требованиям, включая требования по безопасности, а также для выявления отличий между его реально существующими и требуемыми функциями (свойствами).

8.1.2 Функциональное тестирование на модульном уровне реализуется в рамках процесса динамического анализа, предусмотренного ГОСТ Р 56939-2024 (пункт 5.11).

Функциональное тестирование на интеграционном уровне реализуется в рамках процесса динамического анализа (пункт 5.11) или функционального тестирования (пункт 5.18), предусмотренных ГОСТ Р 56939-2024, что определяется разработчиком ПО на основании целесообразности.

Функциональное тестирование на системном уровне реализуется в рамках процесса функционального тестирования, предусмотренного ГОСТ Р 56939-2024 (пункт 5.18).

8.1.3 В зависимости от цели функциональное тестирование может быть выборочным (тестирование определенных требований ПО) и полным. В качестве методов тестирования могут рассматриваться методы, основанные на техниках тестирования, например, тестирование на основе моделей с использованием математических методов, тестирование по сценариям, тестирование на основе опыта и другие тестовые практики, определенные в ГОСТ Р 56920-2024, а также тестирование с использованием тестовых оракулов.

Примечания

1 К выборочному функциональному тестированию могут относиться «дымовое» тестирование («Smoke Testing», тестирование критической функциональности), «тестирование критического пути» («Critical Path Testing», тестирование наиболее важных и часто используемых функций ПО), «санитарное тестирование» («Sanity Testing», тестирование только того функционала, который изменился после устранения недостатков). Данный перечень является неисчерпывающим примером.

2 При тестировании на основе моделей формируются формальные или полужформальные спецификации требуемого поведения программы. При тестировании на основе математических методов эти методы используются для планирования, разработки, отбора данных и определения

входных условий, если требуемое поведение программы, пространство ввода или пространство вывода можно описать с помощью формального языка. Формальные языки спецификаций используются для поддержки тестирования на основе моделей.

3 При тестировании по сценариям тест-кейсы фиксируются (например, в системе управления тестированием), после чего могут быть выполнены ручным или автоматизированным способом с помощью инструментов автоматизированного тестирования.

4 Тестирование на основе опыта основывается на знаниях, собранных по итогам предыдущего тестирования, знаниях объекта анализа, предметной области и ПО в целом. Особенность данного вида тестирования состоит в том, что условия тестирования, специфика входных тестовых данных определяется не общими представлениями о функциональности и способах реализации объекта анализа, а на основе прецедентов проблемных ситуаций, возникавших в ходе разработки данного объекта тестирования или схожих программных систем. Примером такого знания является, например, рекомендация проверять корректность выполнения функций в граничных точках допустимых значений параметров и в соседних точках границ.

5 Тестирование с использованием тестовых оракулов можно проводить в рамках любого вида тестирования, если определены критерии корректности тестируемых функций. Оно позволяет автоматически в ходе выполнения тестов оценивать корректность фактических результатов и (или) побочные эффекты функций и формировать вердикт. В простейшем случае тестовый оракул проверяет выполнение одного из требований к объекту анализа. Общая проверка корректности требует выполнения комбинации тестовых оракулов.

8.1.4 Функциональное тестирование может выполняться ручным, автоматизированным или автоматическим способом. Выбор способа тестирования зависит от различных факторов, например, от прогнозируемого количества и частоты повторных запусков тест-кейсов, возможностей инструментов динамического анализа и автоматизации тестирования. Ручное тестирование может выполняться в случаях необходимости тестирования новых функциональных требований в период разработки на их основе автоматических тест-кейсов.

8.1.5 Спецификация функциональных требований должна содержать, как минимум:

- требования к ПО, реализующие функциональные возможности ПО по назначению, с выделением требований, критичных для работоспособности и безопасности ПО;
- требования безопасности ПО.

В спецификацию функциональных требований рекомендуется включать все декларируемые функциональные возможности ПО.

8.1.6 В качестве объекта анализа должны быть определены:

- на системном уровне — ПО в целом или подсистемы ПО;
- на интеграционном уровне — программные компоненты (в т.ч. подсистемы)

ПО и (или) программные модули ПО;

- на модульном уровне - как минимум, программные модули ПО, реализующие требования назначения и требования (функции) безопасности к ПО.

8.1.7 В рамках сбора дополнительных сведений в целях подготовки к функциональному тестированию на модульном и интеграционном уровнях должны быть получены сведения об интерфейсах (API) программных модулей, подлежащих функциональному тестированию, их взаимодействию с другими модулями ПО.

8.1.8 При формировании требований к тестовой среде должны быть учтены следующие требования:

1) для выполнения функционального тестирования на системном уровне:

- о соответствии тестовой среды среде функционирования ПО и ее настройке в соответствии с эксплуатационной документацией ПО;

2) для выполнения функционального тестирования на модульном уровне:

- к сборочной среде с целью формирования тестовой сборки с встроенными датчиками срабатывания ошибок (если такое встраивание допускает среда сборки, см. 6.3.3);

- к среде выполнения для обеспечения ее изоляции (путем применения «заглушек», имитаций).

8.1.9 Для выполнения функционального тестирования на системном уровне должны быть разработаны тест-кейсы и сценарии тестирования для покрытия функциональных требований к ПО согласно спецификации с учетом определенных приоритетов тестирования.

При разработке тест-кейсов необходимо:

- руководствоваться требованиями, предъявляемыми к процессу функционального тестирования ГОСТ Р 56939-2024 (пункт 5.18);

- руководствоваться эксплуатационной документацией на ПО;

- учитывать установленный приоритет тестирования отдельных функциональных требований (функций) ПО и (или) программных модулей.

При разработке тест-кейсов должно выполняться условие: каждый тест-кейс должен вырабатывать один тестовый вердикт (результат проверки реализации требования, на которую был нацелен данный тест-кейс).

П р и м е ч а н и е - Как правило за выработку тестового вердикта отвечает тестовый оракул.

8.1.10 Для выполнения функционального тестирования на модульном уровне должны быть разработаны юнит-тесты, как минимум, для модулей, реализующих функции (требования) безопасности ПО и критичные для работоспособности ПО функциональные возможности.

Предъявляются следующие базовые требования к юнит-тестам:

- атомарность (один тест проверяет только одно действие программного модуля);
- изолированность и независимость (автономность, выполнение в изоляции без зависимости от каких-либо внешних факторов, таких как файловая система или база данных);
- читаемость;
- повторяемость (выполнение теста должно соответствовать его результатам, тест всегда возвращает один и тот же результат при отсутствии изменений) и другие.

8.1.11 Тестовая среда, среда сборки и среда выполнения должны быть настроены с учетом требований, указанных в 8.1.8.

8.1.12 Для выполнения функционального тестирования на модульном уровне (модульное тестирование) должна быть подготовлена тестовая сборка с учетом требований, установленных в пункте 6.3.3.

8.1.13 Функциональное тестирование на модульном уровне рекомендуется выполнять непосредственно разработчикам программного кода, не применяя принцип независимости тестировщиков от разработчиков. Функциональное тестирование на системном уровне — тестировщикам.

8.1.14 Сбор тестового покрытия должен обеспечиваться, как минимум:

- на модульном уровне тестирования — тестового покрытия программного кода для каждого модуля;
- на системном уровне тестирования — тестового покрытия требований (на основании спецификации требований).

8.1.15 При обработке результатов должен быть выполнен анализ фактических результатов функционального тестирования на предмет соответствия ожидаемым результатам, нарушения требований, проверяемых тест-кейсами, в том числе

порождаемых датчиками срабатывания ошибок.

8.1.16 При анализе результатов функционального тестирования необходимо учитывать, что обязательным условием выполнения требований высокого уровня является выполнение соответствующих ему требований низкого уровня.

8.1.17 Тест-кейсы и сценарии тестирования, проверяющие критичные для работоспособности ПО функциональные возможности и требования безопасности ПО, должны быть автоматизированы.

8.2 Тестирование на основе требований: нефункциональное тестирование

8.2.1 Нефункциональное тестирование применяется для проверки соответствия объекта анализа предъявляемым к нему нефункциональным требованиям. К нефункциональным требованиям могут относиться требования доступности и отказоустойчивости, производительности и иные, невыполнение или недостатки реализации которых могут привести к нарушению доступности информации и связанных с этим угроз информационной безопасности.

8.2.2 Виды нефункционального тестирования определяются разработчиком в соответствии с предъявляемыми к ПО нефункциональными требованиями и учетом результатов моделирования угроз ПО. К нефункциональному тестированию относятся:

- тестирование отказоустойчивости, при котором выполняется проверка устойчивости и способности объекта анализа функционировать при условиях, направленных на отказ (например, при возникновении нарушений, опасных ситуаций в окружении или аппаратуре);

- нагрузочное тестирование, при котором выполняется проверка работоспособности объекта анализа при дефиците ресурсов (памяти оперативной или внешней, пропускной способности сети, свободного времени процессора и пр.) или повышенной нагрузке (большое число запросов, большой объем входных данных, повышенная сложность входных данных и пр.) в условиях штатного функционирования;

- тестирование надежности, при котором выполняется проверка возможности объекта анализа выполнять свои функции (включая оценку частоты, с которой происходят отказы) при использовании в установленных условиях в течение заданного периода времени;

- стрессовое тестирование, при котором выполняется проверка работоспособности объекта анализа при экстремальных нагрузках, на границе выхода из строя и за её пределами, то есть во внештатных условиях функционирования;

- иные виды.

8.2.3 Нефункциональное тестирование реализуется на системном уровне в рамках процесса нефункционального тестирования, предусмотренного ГОСТ Р 56939-2024 (пункт 5.19).

8.2.4 Нефункциональное тестирование может выполняться ручным или автоматизированным способом. Выбор способа тестирования зависит от возможностей инструментов динамического анализа и автоматизации тестирования, а также целесообразности применения инструментов.

8.2.5 Спецификация требований должна содержать предъявляемые к ПО нефункциональные требования, среди которых рекомендуется выделять требования, критичные для работоспособности ПО. Приоритет, периодичность и способы тестирования в рамках регулярного нефункционального тестирования рекомендуется устанавливать с учетом критичных для работоспособности ПО требований.

8.2.6 В качестве объекта анализа на системном уровне рекомендуется определять ПО в целом или подсистемы ПО.

8.2.7 При формировании требований к тестовой среде должны быть учтены требования о соответствии тестовой среды среде функционирования ПО и ее настройке в соответствии с эксплуатационной документацией ПО.

8.2.8 Должны быть разработаны тест-кейсы и сценарии тестирования для покрытия нефункциональных требований к ПО согласно спецификации с учетом выделенных приоритетов тестирования.

При разработке тест-кейсов необходимо:

- руководствоваться требованиями, предъявляемыми к процессу нефункционального тестирования ГОСТ Р 56939-2024;

- руководствоваться эксплуатационной документацией на ПО;

- учитывать установленный приоритет тестирования отдельных нефункциональных требований ПО.

При разработке тест-кейсов должно выполняться условие: каждый тест-кейс должен вырабатывать один тестовый вердикт (результат проверки реализации требования, на которую был нацелен данный тест-кейс).

8.2.9 Тестовая среда должна быть настроена с учетом требований, указанных в 8.2.7.

8.2.10 При внедрении и выполнении нефункционального тестирования должно обеспечиваться тестовое покрытие требований на основании спецификации нефункциональных требований

8.2.11 Нефункциональное тестирование должно выполняться с применением принципа независимости тестировщиков от разработчиков.

8.3 Фаззинг-тестирование

8.3.1 Фаззинг-тестирование применяется для выявления уязвимостей объекта анализа путем подачи на вход объекта анализа набора случайных или специально сформированных входных данных, создаваемых вручную или с использованием инструментов динамического анализа (в частности, фаззеров), с последующим отслеживанием возникновения сбоев в его работе (аварийных завершений, утечек памяти).

8.3.2 Фаззинг-тестирование реализуется на модульном или системном уровнях в рамках процесса динамического анализа, предусмотренного ГОСТ Р 56939-2024 (пункт 5.11).

8.3.3 Методы фаззинг-тестирования:

1) в зависимости от способа генерации входных данных:

- мутационное фаззинг-тестирование: генерация входных данных происходит методом «мутаций» входных данных;

- генерационное фаззинг-тестирование: генерация входных данных осуществляется в соответствии с заданным форматом;

П р и м е ч а н и е — Мутации данных заключаются в изменении некоторых участков входных данных (в качестве таких изменений могут осуществляться случайное инвертирование битов, перестановка блоков данных, изменение порядка следования блоков данных и иные).

2) в зависимости от наличия исходного кода, сведений об архитектуре и функциональных возможностях объекта анализа:

- фаззинг-тестирование методом черного ящика (сведения о внутреннем устройстве объекта анализа отсутствуют);

- фаззинг-тестирование методом серого ящика (имеются неполные сведения о внутреннем устройстве объекта анализа);
- фаззинг-тестирование методом белого ящика (имеются полные сведения о внутреннем устройстве объекта анализа).

8.3.4 Фаззинг-тестирование рекомендуется выполнять автоматизированным или автоматическим способом.

8.3.5 При внедрении и на этапе подготовки к регулярному к фаззинг-тестированию в качестве справочного материала рекомендуется использовать спецификацию функциональных и нефункциональных требований, а также функциональные сценарии тестирования и тест-кейсы.

8.3.6 В качестве объекта анализа должны быть определены:

- на системном уровне (если применимо) - ПО в целом или отдельные компоненты (подсистемы) ПО;
- на модульном уровне - как минимум, программные модули ПО, составляющие поверхность атаки ПО и реализующие требования (функции) безопасности к ПО.

П р и м е ч а н и е - Данные объекты должны быть включены в перечень целей для исследований безопасности ПО, формируемых в рамках процесса моделирования угроз и описания поверхности атаки согласно требованиям ГОСТ Р 56939-2024 (пункт 5.7).

Также в качестве объектов анализа на модульном уровне рекомендуется рассматривать программные модули, реализующие критические для работоспособности и производительности ПО функции (требования). Такие модули могут быть определены на основании анализа спецификации требований к ПО.

8.3.7 Для каждого объекта анализа должны быть определены фаззинг-цели — конкретные участки кода (функции) и интерфейсы программных модулей, которые подлежат фаззинг-тестированию.

Рекомендуется выделять фаззинг-цели, реализующие обработку и (или) преобразование входных данных (парсеры, анализаторы данных), которые может сформировать потенциальный нарушитель, реализующие функции идентификации и аутентификации пользователей, функции управления доступом, иные функции и свойства, определяемые разработчиком ПО как критерии выбора фаззинг-целей. Критерии выбора фаззинг-целей должны быть задокументированы в организационно-распорядительной документации разработчика ПО.

8.3.8 Должен быть установлен приоритет тестирования фаззинг-целей. Критерии выбора и приоритет выполнения тестирования фаззинг-целей должны быть определены в организационно-распорядительной документации разработчика ПО.

8.3.9 В рамках сбора дополнительных сведений в целях подготовки к фаззинг-тестированию:

1) должны быть получены следующие дополнительные сведения о программных модулях, подлежащих тестированию:

- об используемых языках программирования;
- об интерфейсах (API) программных модулей (наименование и назначение API, способах и протоколах взаимодействия, о типах обрабатываемых данных, доступности интерфейса потенциальным нарушителям);

2) определены точки входных воздействий и наблюдения поведения программы;

3) определена среда функционирования ПО, в которой будет выполняться фаззинг-тестирование.

При определении среды функционирования может быть выбрана только одна среда (если сред функционирования две и более), обеспечивающая наиболее эффективное выполнение фаззинг-тестирования. В случае технической невозможности выполнения фаззинг-тестирования в среде функционирования ПО, может быть определена среда, отличная от среды функционирования ПО.

П р и м е ч а н и е — Техническая невозможность может быть обусловлена отсутствием фаззеров, функционирующих в среде функционирования ПО. Например, в среде операционных систем, функционирующих на базе процессорной архитектуры, не поддерживаемой фаззерами.

Для фаззинг-целей необходимо определить сигнатуру функции, формат обрабатываемых данных, классы эквивалентности ввода и граничные значения, имеющихся ограничениях на количество и частоту вызовов, а также сопоставить каждую фаззинг-цель и с программным модулем, в состав которого она входит.

8.3.10 Особенности при формировании требований к тестовой среде аналогичны установленным в пункте 8.1.8 с допущением, что среда функционирования может быть только одна (если их две и более) или отличаться от декларированной в эксплуатационной документации среды функционирования, если это определено на этапе сбора дополнительных сведений.

8.3.11 В рамках этапа по разработке тест-кейсов и тестовых данных должны быть сформированы наборы входных данных.

При формировании наборов входных данных необходимо обеспечить:

- валидность (данные не должны приводить к аварийному завершению программы);
- покрытие различных ветвей программного кода;
- исключение дубликатов;
- минимальность размера файлов данных;
- представительность (включить различные версии, режимы, опции формата файлов входных данных, а также граничные значения входных данных).

Начальный корпус входных данных необходимо минимизировать средствами фаззера.

Источники, используемые при формировании наборов входных данных:

- тестовая документация и тест-кейсы функционального тестирования, автоматические и юнит-тесты;
- сценарии применения ПО согласно эксплуатационной и иной документации;
- открытые наборы данных;
- ранее сформированные данные;
- сгенерированные данные;
- синтетические данные.

При использовании фаззеров, использующих коллекции входных данных, для каждой фаззинг-цели должны быть сформированы тестовые коллекции входных данных, подлежащие дальнейшим мутациям (на основе публично доступных коллекций, с использованием генераторов коллекций, с использованием инструментов автоматического определения и формирования коллекций данных, экспертным способом). Коллекции должны вызывать использование различных функциональных возможностей интерфейса программного модуля, подлежащего фаззинг-тестированию (фаззинг-цели).

При использовании фаззеров, не поддерживающих коллекции входных данных, должны быть сформированы правила порождения входных данных для каждого интерфейса программного модуля, подлежащего фаззинг-тестированию (фаззинг-цели). Правила порождения должны вызывать использование различных функциональных возможностей тестируемого интерфейса.

При использовании фаззеров, поддерживающих работу со словарями, необходимо сформировать и применить словари лексем за исключением случаев, определяемых разработчиком ПО.

Разработчик ПО должен сохранять сформированные наборы данных.

8.3.12 В рамках этапа по разработке тест-кейсов и тестовых данных должны быть разработаны функции-обертки для передачи данных в аргументы функций или на входные интерфейсы объектов анализа.

Функции-обертки обеспечивают вызов тестируемых участков кода (целевой функции), конвертируя данные во входные параметры тестируемых функций в соответствии со спецификациями интерфейсов (протоколов), типов данных и контекста выполнения тестируемых функций в составе ПО.

Рекомендуемые требования к функциям-оберткам:

- обеспечение приема входных данных от фаззера и преобразование их в аргументы целевой функции;
- возвращение унифицированного формата ответа;
- перенаправление (или закрытие) потоков вывода, не влияющих на результаты тестирования;
- перехват ожидаемых (валидных с точки зрения логики) исключений и аварийное завершение при сбоях;
- содержание только минимально необходимой логики и минимального объема выполняемых операций;
- возможность многократного последовательного вызова в рамках одного процесса и обеспечения независимости итераций;
- воспроизведение сценариев и результатов тестирования при одинаковых условиях и входных данных.

При разработке функций-оберток могут использоваться инструменты, позволяющие в автоматизированном режиме осуществить генерацию оберток на основе исходного кода ПО.

Разработку функций-оберток рекомендуется выполнять непосредственно разработчикам программного кода тестируемых модулей (фаззинг-целей).

8.3.13 Фаззинг-тестирование может выполняться в отношении специально подготовленных синтетических целей посредством выделения из исходного кода модулей ПО тех участков, которые непосредственно реализуют обработку входных

данных. Необходимость и целесообразность разработки синтетических целей (синтетические бинарных модулей) определяется разработчиком ПО.

Если такая необходимость определена, в рамках этапа по разработке тест-кейсов и тестовых данных должны быть разработаны синтетические бинарные модули.

Для создания синтетического бинарного модуля после выделения интересующих участков кода необходимо:

- разработать функции-обертки;
- изолировать код для фаззинг-тестирования, составив синтетический бинарный модуль из выделенных участков и тестовых дублеров («тоск-объекты», «заглушки», имитации);
- выполнить сборку модуля с встраиванием датчиков срабатываний ошибок.

Рекомендованные требования к синтетическим бинарным модулям:

- изоляция кода путем применения тестовых дублеров («тоск-объектов», «заглушек», имитаций);
- взаимодействие с фаззером (прием и обработка входных данных, поступающих от фаззера);
- вызов целевой тестируемой функции (фаззинг-цели);
- корректное и своевременное освобождение выделенных ресурсов при завершении работы;
- возможность сборки (компиляции) с применением санитайзеров;
- отсутствие сторонних тестовых функций и исполняемых автотестов.

8.3.16 Разработчик ПО должен зафиксировать параметры сборки каждого программного модуля (фаззинг-цели), в отношении которого выполнялось фаззинг-тестирование, принципы формирования коллекции, правил, словарей, достигнутое тестовое покрытие.

8.3.17 Разработанные тест-кейсы и наборы данных должны использоваться при выполнении повторного и регрессионного фаззинг-тестирования.

8.3.18 В целях повышения эффективности фаззинг-тестирования при регрессионном тестировании рекомендуется использовать новые тест-кейсы и наборы данных или адаптировать имеющиеся.

8.3.19 Тестовая среда, среда сборки и среда выполнения должны быть настроены с учетом требований, указанных в 8.3.6. Для выполнения фаззинг-

тестирования на модульном уровне должна быть подготовлена тестовая сборка с учетом требований 6.3.3.

В рамках подготовки тестовой среды должны быть обеспечена возможность запуска как тестируемых объектов анализа, так и функций-оберток.

8.3.20 При внедрении и выполнении фаззинг-тестирования должен обеспечиваться сбор тестового покрытия программного кода и определены критерии завершения фаззинг-тестирования.

П р и м е ч а н и е — Возможные критерии завершения фаззинг-тестирования:

1) количество найденных путей или достигнутого покрытия:

- для фаззеров, подсчитывающих число найденных уникальных путей, — число найденных уникальных путей не менее чем в 2 раза превышает число стартовых образцов входных данных, поданных на вход фаззеру;

- для фаззеров, подсчитывающих достигнутое покрытие, — процент покрытия, достигнутого фаззером, не менее чем в 2 раза превышает процент покрытия, достигаемый в результате подачи на вход фаззеру стартовых образцов входных данных;

2) тестовое покрытие программного кода, определяемое на основании достижения покрытия по строкам исходного кода или по базовым блокам не менее установленного числа процентов;

3) продолжительность фаззинг-тестирования, определяемая на основании установленного числа часов, в течении которых фаззер не находит новые пути и (или) отсутствует прирост покрытия программного кода при условии, что на вход подаются уникальные мутированные данные.

8.3.22 Результаты фаззинг-тестирования должны содержать:

- сведения о результатах работы инструментов фаззинг-тестирования (длительность проведения фаззинг-тестирования, количество аварийных завершений работы ПО, количество найденных путей выполнения и иные);

- сведения о срабатываниях инструментальных средств;

- диагностические данные: дампы аварийных завершений, диагностическая информация от инструментальных датчиков срабатывания ошибок, ошибки в поведении программы (объекта анализа);

- наборы данных, подача которых приводит к ошибкам в поведении программы (объекта анализа) или аварийному завершению;

- достигнутое структурное покрытие по строкам исходного кода или по базовым блокам объектов анализа.

8.3.23 Анализ результатов фаззинг-тестирования должен включать:

- определение достигнутого покрытия по строкам исходного кода или базовым блокам исполняемого кода для каждого тестируемого программного модуля;

- анализ обнаруженных ошибок в поведении (аварийных завершений, «зависаний»), определение их актуальности;
- определение типа актуальных ошибок и их критичности;
- проверку непокрытых в ходе тестирования участков кода с целью обнаружения программных закладок и других потенциально опасных элементов, а также с целью оценки необходимости разработки новых тест-кейсов или доработки существующих.

При анализе ошибок необходимо учитывать, что найденные аварийные завершения могут указывать как на ошибки в исходной программе, так и на некорректно подготовленные синтетическую цель или функцию-обертку.

8.3.24 Если при проведении фаззинг-тестирования были выявлены механизмы, препятствующие проведению исследований, программные модули или участки кода, в которых они обнаружены, рекомендуется подвергнуть дополнительному анализу, например, в ходе экспертизы исходного кода ПО.

8.4 Динамический анализ помеченных данных

8.4.1 Динамический анализ помеченных данных применяется для выявления ошибок поведения программы, которые могут привести к утечке конфиденциальных сведений.

8.4.2 Динамический анализ помеченных данных применяется для выявления следующего поведения программы, которое трактуется, как ошибка.

В программе определены операторы, на выполнение которых потенциальный нарушитель не должен иметь возможностей повлиять. Помечаются входные данные, которые потенциальный нарушитель контролирует (имеет возможность формировать произвольным образом). В ходе анализа помеченные данные достигают определенных операторов и влияют на их выполнение.

В программе определены защищаемые данные, которые помечаются. В ходе анализа помеченные данные достигают операторов, в которых осуществляется вывод помеченных данных в открытые каналы передачи информации.

П р и м е ч а н и е — Данным из определенных источников присваивается специальная пометка, выполнение каждого оператора программы сопровождается распространением пометки на вычисляемые данные, если они зависят от данных, имеющих пометку.

8.4.3 Динамический анализ помеченных данных реализуется на системном уровне в рамках процесса динамического анализа, предусмотренного ГОСТ Р 56939-2024 (пункт 5.11).

8.4.4 Реализация динамического анализа помеченных данных определяет:

- правила пометки данных (источников помеченных данных);
- правила распространения пометок при выполнении операторов программы;
- критерии, когда поведение программы должно трактоваться как ошибка поведения.

8.4.5 Динамический анализ помеченных данных может быть реализован с использованием вспомогательных методов:

- динамического бинарного инструментирования кода;
- программной полносистемной эмуляции;
- инструментирования кода программы компилятором во время компиляции.

Не допускается реализация динамического анализа помеченных данных на уровне языка программирования средствами инструментирования исходного кода.

Рекомендуется применять реализацию динамического анализа помеченных данных, обеспечивающую анализ разрабатываемого ПО совместно со средой функционирования ПО.

П р и м е ч а н и е — Динамический анализ помеченных данных средствами интерпретации языка программирования, например, Ruby и Perl, малоэффективен и не рекомендуется к использованию.

8.4.6 Динамический анализ помеченных данных может быть реализован как непосредственно во время выполнения программы, так и путем анализа трасс и профилей.

8.4.7 В рамках сбора дополнительных сведений об объекте анализа дополнительно должны быть определены:

- 1) типы защищаемых данных (пароли, определенные файлы, данные, связанные с пользовательским вводом и иные), которые подлежат отслеживанию (помеченные данные);
- 2) источники ввода помеченных данных: программные модули, реализующие входные интерфейсы для помеченных данных;
- 3) правила распространения пометок при выполнении операторов программы.

П р и м е ч а н и е - Правила распространения пометок могут быть определены на уровне байт или отдельных битов. Определение правил распространения пометок на уровне машинных слов — недопустимо.

8.4.8 На этапе разработки тест-кейсов и сценариев тестирования разработчик ПО должен определить сценарии функционирования ПО, в которых осуществляется обработка помеченных данных.

П р и м е ч а н и е - При выборе сценариев следует отдавать предпочтение сценариям, затрагивающим порождение (модификацию) и (или) управление помеченными данными, а также предполагающим взаимодействие с программными компонентами, расположенными на различных узлах сети.

Сценарии тестирования и тест-кейсы должны быть основаны на результатах анализа функциональных возможностей ПО, в которых выполняется ввод и обработка помеченных данных.

8.4.9 На этапе подготовки тестовой среды должна быть подготовлена тестовая сборка, позволяющая провести динамический анализ помеченных данных.

Если разработчик ПО применяет средства динамического бинарного инструментирования кода и инструментирования кода программы компилятором во время компиляции, то должна быть проведена оценка влияния внедряемого инструментального кода на поведение программы и предприняты действия по исключению искажений поведения программы.

8.4.10 В ходе динамического анализа помеченных данных необходимо контролировать сетевую активность и взаимодействие ПО с носителями информации (например, посредством анализа системных вызовов) с целью выявления недеklarированных возможностей и утечек данных.

8.4.11 Качество (точность) динамического анализа помеченных данных определяется на основании следующих критериев:

- недостаточная помеченность – когда зависимость между данными имеется, но применяемый в реализации набор правил распространения пометок не приводит к пометке зависимых данных;
- избыточная помеченность – когда данные не зависят друг от друга, но применяемый в реализации набор правил распространения пометок приводит к пометке независимых данных.

П р и м е ч а н и е - Качество реализации динамического анализа помеченных данных оценивается на наборе тестов.

8.5 Тестирование на отказ

8.5.1 Тестирование на отказ заключается в проверке возможности привести объект анализа в аварийное состояние путем искусственного внедрения ошибок или

имитации сбоев.

Примечание — Аварийное состояние характеризуется сбоем, в результате которого объект тестирования либо не может выполняться (аварийно завершается), либо выполнение становится безрезультативным (например, при заикливании).

8.5.2 Тестирование на отказ реализуется на системном уровне в рамках процесса нефункционального тестирования согласно ГОСТ Р 56939-2024 (пункт 5.19).

8.5.3 Тестирование на отказ может проводиться самостоятельно или совместно с другими видами нефункционального тестирования, определенными разработчиком ПО, такими как тестирование отказоустойчивости, нагрузочное тестирование, тестирование надежности, стрессовое тестирование.

8.5.4 Тестирование на отказ может выполняться ручным или автоматизированным способом.

8.5.5 В рамках сбора дополнительных сведений должны быть определены потенциальные точки отказа ПО на основании анализа архитектуры, зависимостей между сервисами (анализ архитектуры и выявление рисков).

8.5.6 При тестировании на отказ могут проверяться те же функциональные или нефункциональные возможности (свойства) объекта анализа, что и при функциональном или нефункциональном тестировании, но сценарии тестирования предполагают целенаправленное создание ситуации, при которой взаимодействующие с объектом анализа модули, компоненты, устройства ведут себя некорректным образом.

8.5.7 В рамках подготовки тестовой среды должно быть обеспечено ее соответствие среде эксплуатации, выполнены настройки в соответствии с эксплуатационной документацией.

8.5.8 При подготовке тест-кейсов и сценариев тестирования должны быть описаны тип сбоя, условия его возникновения, ожидаемый результат и критерии успешного прохождения (разработать матрицу тестовых сценариев). Тест-кейсы на отказ включают использование не вполне корректных входных данных.

8.5.9 Полнота тестирования на отказ должна контролироваться при помощи измерения покрытия тест-кейсами множества разнообразных сценариев, которые могут возникнуть в окружении или входных данных тестируемого объекта, включая разные элементы интерфейсов и структур входных данных, в которых могут возникать нарушения, а также разные участки программы, в которых эти проблемы могут

воздействовать на тестируемый объект.

8.6 Тестирование на проникновение

8.6.1 Тестирование на проникновение применяется для выявления и подтверждения актуальности уязвимостей ПО путем имитации действий потенциального нарушителя.

8.6.2 Тестирование на отказ реализуется на системном уровне в рамках процесса нефункционального тестирования согласно ГОСТ Р 56939-2024 (пункт 5.19).

8.6.3 Тестирование на проникновение может выполняться ручным или автоматизированным способом.

8.6.4 Тестирование на проникновение осуществляется в соответствии с ГОСТ Р 58143-2018 и другими национальными стандартами или нормативно-правовыми документами в области защиты информации и в область стандартизации настоящего стандарта не входит.

8.7 Динамическая верификация

8.7.1 Динамическая верификация применяется для проверки соответствия поведения программы (объекта анализа) поведению, предусмотренному его формальной моделью.

Для ПО, классифицируемого как СЗИ, реализующее политики управления доступом (далее - ПО СЗИ), динамическая верификация применяется для проверки соответствия функционирования ПО СЗИ формальной модели управления доступом, разработанной в соответствии с критериями и рекомендациями ГОСТ Р 59453.1-2021, ГОСТ Р 59453.2-2021 и ГОСТ Р 59453.3-2025.

8.7.2 Динамическая верификация реализуется на системном уровне в рамках процесса функционального тестирования согласно ГОСТ Р 56939-2024 (пункт 5.18).

8.7.3 Анализ поведения программы (объекта анализа) сводится к построению отображения его входов и выходов в интерфейсы модели и последующему анализу (или анимации) модели с полученными входными воздействиями и формированию вывода о том, нарушают ли полученные фактические результаты требования модели или нет.

8.7.4 Динамическая верификация ПО СЗИ осуществляется в соответствии

с ГОСТ Р 59453.4-2025 и в область стандартизации настоящего стандарта не входит.

8.8 Вспомогательные методы динамического анализа

8.8.1 Снятие трасс выполнения программы

Метод применяется в технологиях динамического анализа в целях получения детальной информации о ходе выполнения программы. Собираемая информация показывает последовательность выполнения программы на уровне машинных команд или конструкций исходного языка программирования, а также любые иные события, связанные с выполнением программы. Собранные данные используются для проведения дальнейшего анализа программы.

При снятии трасс замедление в работе программы может достичь критического уровня, при котором поведение программы меняется и искажает собираемую информацию. Для преодоления данного эффекта рекомендуется использовать технологию двухэтапного снятия трасс:

а) на первом этапе фиксируются все внешние для программы недетерминированные события (вход пользователя, сетевой вход, прерывания) с приемлемым уровнем замедления;

б) на втором этапе осуществляется детерминированное воспроизведение программы по начальному состоянию и журналу записанных внешних событий, во время выполнения снимается трасса в необходимом объеме с учетом применения записанных внешних событий.

8.8.2 Инструментирование кода

Инструментирование представляет собой трансформацию программы внесением в нее инструментального кода, реализующего функции некоторого программного инструмента (например, встраивание в программу датчиков срабатываний ошибок).

Инструментирование может выполняться:

- по исходным текстам;
- по бинарному (в том числе промежуточному) представлению.

Инструментирование может быть реализовано как статически, до запуска программы (при сборке), так и динамически, во время исполнения кода программы.

Метод применяется в технологиях динамического анализа с целью извлечения информации о ходе выполнения программы, получения диагностической информации о состоянии программы, в том числе при отладке.

Анализ полученной информации может проводиться как непосредственно по ходу выполнения программы, так и после ее выполнения. При этом важно учитывать, что внесение изменений в код программы может повлиять на ее поведение и быть причиной появления недостатков, которые отсутствуют в немодифицированном коде.

8.8.3 Динамическая бинарная трансляция

Динамическая бинарная трансляция представляет собой трансформацию кода выполняющейся программы, когда входом и выходом (результатом трансляции) трансформации является исполняемый код. Как правило, динамическая бинарная трансляция выполняется в целях обеспечения выполнения кода одной процессорной архитектуры на процессоре другой архитектуры. Трансляция с одновременным инструментированием является основой для других технологий динамического анализа, например, для динамического анализа помеченных данных.

8.8.4 Динамическое символьное выполнение

Динамическое символьное выполнение – метод анализа программ, когда конкретные значения части или всех переменных программы заменяются символьными величинами (формулами). Выполнение операций над символьными переменными приводит к выработке не конкретных величин, а уравнений и неравенств над свободными символьными переменными и константами. Полученная математическая система уравнений неравенств передается специальной программе – решателю задачи выполнимости формул в теориях.

Динамическое символьное выполнение применяется в технологиях динамического анализа с целью увеличения покрытия программного кода и поиска ошибок.

В определенной точке программы происходит разделение траекторий выполнения и вместо одного множества значений символьных данных создается два: в первое множество добавляется ограничение на символьные значения, чтобы условие перехода было истинно, во второе множество — наоборот, включается ограничение, описывающее ложность условия. Каждая точка программы на некоторой траектории выполнения характеризуется набором ограничений на значения

символьных данных. Для определения достижимости интересующих точек программы необходимо разрешить систему ограничений.

Прохождение динамического символьного выполнения через ветвление в коде программы приводит к независимому рассмотрению каждого пути выполнения и, как следствие, трансформации системы формул в две независимых системы с добавлением к ним предиката соответствующего пути выполнения. Такая особенность динамического символьного выполнения приводит к экспоненциальному росту количества рассматриваемых путей выполнения и накладывает ограничение на применение технологии в отношении объектов анализа, состоящего из большого объема кода.

8.8.5 Динамическое профилирование

Метод применяется в технологиях динамического анализа в целях сбора и анализа количественных характеристик выполнения программы (таких как время работы функций, количество вызовов, использование памяти, загрузка процессора, частота операций ввода-вывода) при определенных событиях или через равные промежутки времени выполнения.

Результатом динамического профилирования является профиль (набор характеристик) выполнения, используемый для оценки производительности, выявления «узких мест» и оптимизации программы.

8.8.6 Перехват потоков выполнения программы

Метод применяется в технологиях динамического анализа в целях наблюдения, фильтрации, модификации или подмены поведения программы (функций, методов, системных вызовов, событий) в момент выполнения.

Метод заключается во внедрении пользовательского кода в стандартный поток выполнения программы таким образом, что при наступлении заданного события (вызов функции, доступ к памяти, системный вызов, обработка сообщения) управление временно передаётся перехватывающей процедуре. Перехватывающая процедура может анализировать, изменять, блокировать или журналировать переданные данные, а также инициировать вызов исходной (целевой) функции.

Реализация перехвата включает следующие этапы:

- идентификация целевой точки перехвата (адрес функции, системный вызов, обработчик прерывания, метод класса);
- замещение оригинального кода или таблицы адресов таким образом, чтобы управление передавалось перехватывающей функции;

- выполнение перехватывающей функции, которая при необходимости вызывает оригинальный целевой код после или до выполнения собственных действий.

Одним из вариантов перехвата управления также является имитация отказов при вызове к библиотечным функциям или сервисами операционной системы. Эта техника известна как внедрение отказов («fault injection»). Она используется для проверки устойчивости ПО к возможным отказам программ или аппаратуры.

9 Требования к инструментам динамического анализа

9.1 Общие функциональные требования, которые должны обеспечивать инструменты динамического анализа.

9.1.1 Возможность адаптации (доработки) под особенности среды разработки ПО, в том числе тестовой и сборочной среды, что достигается одним из следующих способов:

- в отношении инструментов осуществляется техническая поддержка (регулярный выпуск обновлений, в том числе направленных на устранение ошибок, совершенствование функциональных возможностей);
- инструменты свободно распространяются в исходных кодах, имеющих возможность адаптации (доработки);
- инструменты являются собственной разработкой разработчика ПО.

9.1.2 Детектирование (в зависимости от назначения инструмента) следующих типов ошибок:

- ошибки, выявляемые датчиками срабатывания ошибок;
- аварийное завершение, непредусмотренное требованиями к объекту анализа;
- ошибки, выявляемые тестовыми оракулами (при использовании);
- ошибки непроверенного использования ввода пользователя или ввода данных по разным интерфейсам (например, по сетевым);
- ошибки работы с памятью (например, использование некорректного указателя, выход за пределы выделенной памяти);
- ошибки использования неинициализированной памяти;
- ошибки целочисленного переполнения.

9.1.3 Возможность выполнения анализа в среде функционирования ПО (при этом допускается несоответствие требованию в части фаззеров, так как фаззинг-тестирование может выполняться в среде, отличной от среды функционирования ПО).

9.1.4 Хранение результатов запуска.

9.1.5 Присвоение каждому предупреждению идентификатора, с помощью которого его можно однозначно найти в результатах анализа.

Примечание — Для хранения результатов могут использоваться базы данных и другие сторонние системы.

9.1.6 Возможность интеграции с системами, используемыми в среде разработки (номенклатура определяется разработчиком ПО, например, с системами управления требованиями, отслеживания ошибок, системой непрерывной интеграции и иными).

9.1.7 Обеспечение тестирования объектов анализа, функционирующих в привилегированном режиме (при тестировании ядра операционной системы).

9.2 Функциональные требования, которые должны обеспечивать инструменты (набор инструментов) фаззинг-тестирования (фаззеры).

9.2.1 Реализация следующих методов фаззинг-тестирования:

- мутационное фаззинг-тестирование с поддержкой мутаций, выполняемых с помощью генетических алгоритмов;
- генерационное фаззинг-тестирование;
- фаззинг-тестирование с применением динамического символьного выполнения.

9.2.2 Обнаружение ситуаций нештатного функционирования ПО (аварийных завершений, «зависаний»). В случае, если нештатная ситуация вызвана возникновением критической ошибки, фаззер также должен определять конкретный тип ошибки.

9.2.3 Поддержка режима работы, в котором различные входные данные исследуются параллельно.

9.3 Функциональные требования, которые должны обеспечивать инструменты разработки, генерации и исполнения тест-кейсов.

9.3.1 Сбор результатов тестирования.

9.3.2 Детерминированное выполнение тестов и воспроизведение эффектов, которые наблюдаются при тестировании.

9.3.3 Возможность мониторинга и отладки тестов совместно с объектом анализа при использовании ручных технологий создания тестов.

10 Требования к специалистам

10.1 Для обеспечения процессов динамического анализа должны привлекаться специалисты, участвующие в тестировании ПО и обладающие следующими знаниями и умениями:

- знать языки программирования, использованные при реализации анализируемого ПО;
- знать основы и технологии программной инженерии, жизненного цикла программного обеспечения;
- знать базовые принципы устройства операционных систем и основные положения набора стандартов POSIX;
- знать основные сетевые протоколы и сервисы;
- знать функциональное назначение и архитектуру ПО, используемые инструментальные средства разработки и отладки ПО;
- уметь работать с документацией на ПО и программы из его состава для определения функциональных возможностей и формирования спецификации функциональных требований.

10.2 Для обеспечения функционального тестирования должны привлекаться специалисты, участвующие в тестировании ПО и обладающие в дополнение к требованиям 10.1 следующими знаниями и умениями:

- уметь проводить анализ сформированных инструментами функционального тестирования данных и находить ошибки в поведении программы, обнаруженные при тестировании;
- уметь разрабатывать ручные тесты и готовить исходные данные для средств автоматической генерации тестов;
- уметь работать со средствами сбора тестового покрытия.

10.3 Для обеспечения фаззинг-тестирования должны привлекаться специалисты, совмещающие компетенции по разработке ПО с компетенциями в области обеспечения информационной безопасности, обладающие в дополнение к требованиям 10.1 следующими знаниями и умениями:

- знать базовые алгоритмы и структур данных, базовые положения теории сложности алгоритмов;
- знать основы безопасности ПО;

- уметь определять фаззинг-цели;
- уметь выполнять анализ предупреждений инструментов динамического анализа, для выявленных ошибок — оценивать их влияние на безопасность ПО, выделять критические ошибки и уязвимости кода;
- уметь разрабатывать фаззинг-обертки;
- уметь формировать наборы и коллекции входных данных для фаззинг-тестирования, создавать синтетические программы для подготовки синтетических бинарных модулей (если фаззинг-тестирование использует такие модули).

10.4 Для автоматизации процесса выполнения динамического анализа должны привлекаться специалисты, совмещающие компетенции по разработке ПО с компетенциями по информационно-технологическому обслуживанию (системному администрированию), которые должны обладать в дополнение к требованиям 10.1 следующими знаниями и умениями:

- уметь оценивать временные и вычислительные ресурсы, необходимые для выполнения динамического анализа;
- уметь создавать инфраструктуру для проведения анализа и тестирования.

10.5 Рекомендуется привлекать специалистов, имеющих опыт проведения динамического анализа либо прошедших программы повышения квалификации, связанные с рассматриваемыми в стандарте технологиями динамического анализа.

Приложение А
(рекомендуемое)

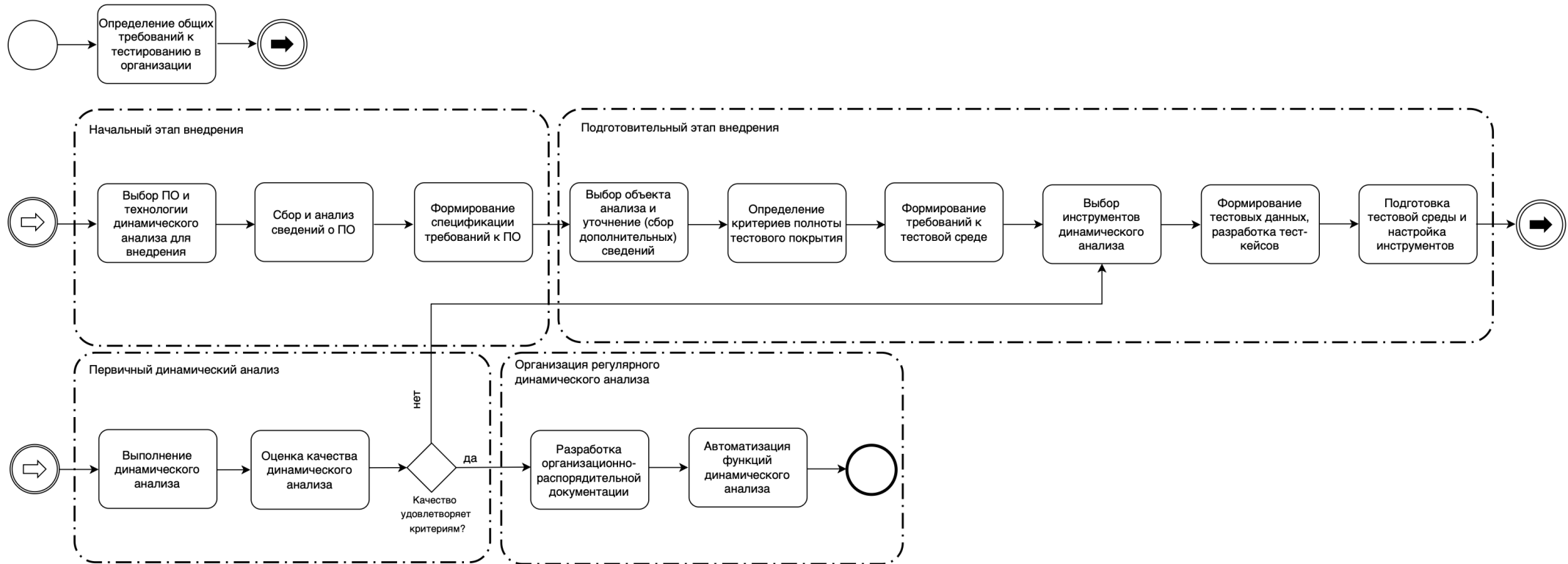
Применение технологий динамического анализа в процессах разработки безопасного программного обеспечения, определенных в ГОСТ Р 56939-2024, с учетом уровней тестирования

Процессы по ГОСТ Р 56939-2024	Технологии динамического анализа	Тестирование на основе требований: функциональное тестирование	Тестирование на основе требований: нефункциональное тестирование	Фаззинг-тестирование	Динамический анализ помеченных данных	Тестирование на отказ	Тестирование на проникновение	Динамическая верификация
Динамический анализ кода программы (пункт 5.11 ГОСТ Р 56939-2024)		Модульный уровень		Модульный уровень Системный уровень	Системный уровень			
Функциональное тестирование (пункт 5.18 ГОСТ Р 56939-2024)		Интеграционный уровень Системный уровень						Системный уровень
Нефункциональное тестирование (пункт 5.19 ГОСТ Р 56939-2024)			Системный уровень			Системный уровень	Системный уровень	

Приложение Б

(справочное)

Диаграмма процесса внедрения технологий динамического анализа



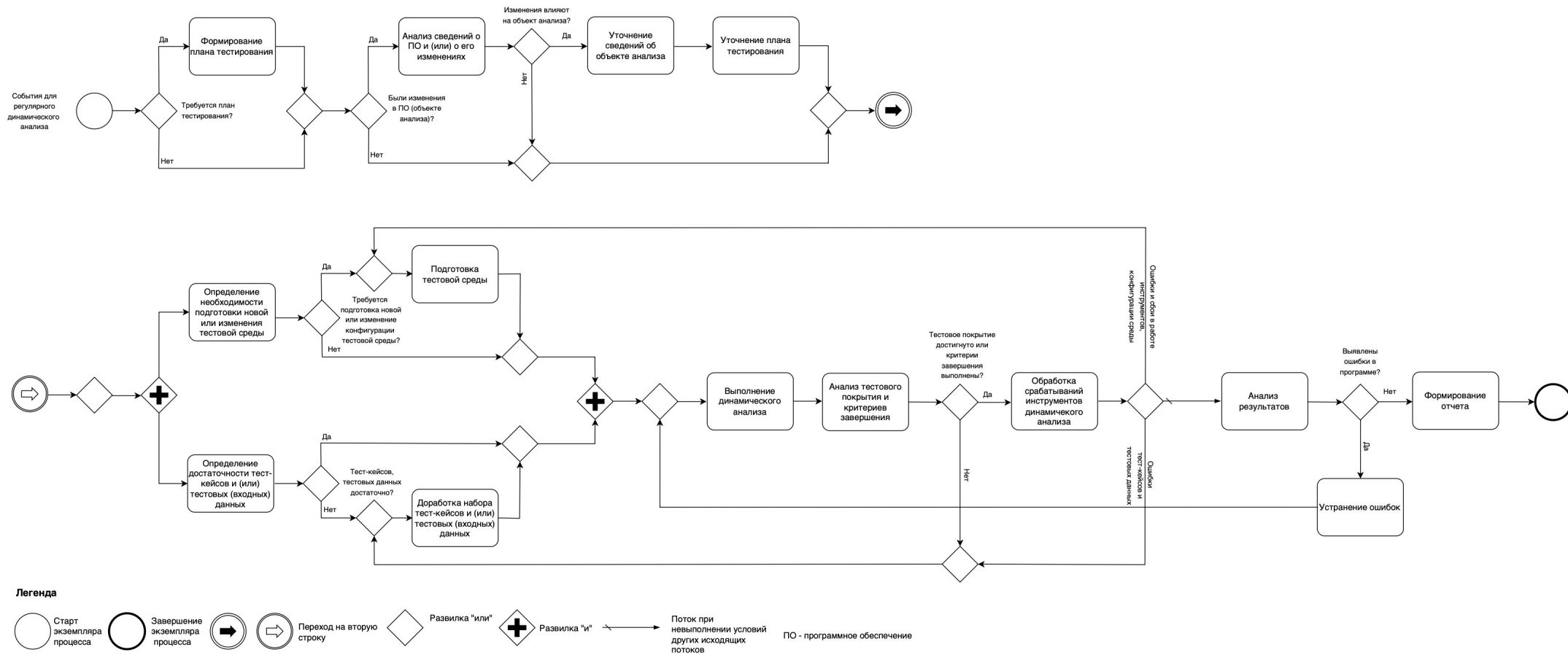
Легенда: ПО - программное обеспечение



Приложение В

(справочное)

Диаграмма процесса выполнения регулярного динамического анализа



УДК 004.056.053

ОКС 35.030:35.080

Ключевые слова: безопасное программное обеспечение, динамический анализ, фаззинг-тестирование, функциональное тестирование, модульное тестирование, системное тестирование, регрессионное тестирование, тестирование на отказ
