МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Искусственного Интеллекта
Базовая кафедра БК252 - информационной безопасности.

ПРАКТИЧЕСКАЯ РАБОТА

По дисциплине
«Методы верификации»

Тема практической работы
«Методы динамического анализа программного обеспечения.»

Студенты группы ККСО-03-19	Николенко В.О. Воеводин К.А.
Руководитель практической работы	Коробов Д. В.

(подпись)

Москва – 2023

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Методы динамического анализа программного обеспечения (Dynamic Software Analysis) позволяют изучать поведение программы во время ее выполнения. Эти методы используются для тестирования, отладки, профилирования и обнаружения ошибок в программном обеспечении.

2. ТЕОРИЯ

2.1. Виды динамического анализа.

Существует несколько методов динамического анализа программного обеспечения:

1. Тестирование черного ящика (Black Box Testing) - метод, при котором программа тестируется без знания ее внутренней структуры. Тестирование черного ящика может быть автоматизировано с помощью специальных инструментов, которые генерируют тестовые данные и анализируют результаты выполнения программы.

2. Тестирование белого ящика (White Box Testing) - метод, при котором программа тестируется с знанием ее внутренней структуры. Тестирование белого ящика может быть автоматизировано с помощью инструментов, которые анализируют код программы и генерируют тестовые данные на основе этого анализа.

3. Профилирование (Profiling) - метод, при котором изучается поведение программы во время ее выполнения с целью определения узких мест и оптимизации производительности. Профилирование может быть автоматизировано с помощью специальных инструментов, которые анализируют производительность программы и выдают отчеты о ее работе.

4. Мониторинг (Monitoring) - метод, при котором изучается поведение программы в реальном времени с целью обнаружения ошибок и проблем. Мониторинг может быть автоматизирован с помощью инструментов, которые анализируют данные, получаемые во время выполнения программы, и выдают предупреждения о возможных проблемах.

5. Дебаггинг (Debugging) - метод, при котором изучается поведение программы во время ее выполнения с целью обнаружения и исправления ошибок. Дебаггинг может быть автоматизирован с помощью инструментов, которые позволяют отслеживать выполнение программы и находить места, где происходят ошибки.

Каждый из методов динамического анализа программного обеспечения имеет свои преимущества и недостатки, и выбор метода зависит от конкретной задачи, которую нужно решить.

2.2. Тестирование чёрного ящика.

Тестирование черного ящика проводится без знания внутренней структуры программы. Для этого используются различные методы, такие как функциональное тестирование, тестирование граничных значений, тестирование случайных значений и др.

Функциональное тестирование проверяет соответствие функциональных требований программы. Например, если программа должна складывать два числа, то функциональное тестирование проверит, что программа действительно складывает два числа и выдает правильный результат.

Тестирование граничных значений проверяет поведение программы на крайних значениях входных данных. Например, если программа должна обрабатывать числа от 1 до 100, то тестирование граничных значений проверит поведение программы при вводе чисел 1 и 100.

Тестирование случайных значений проверяет поведение программы на случайных входных данных. Для этого используются специальные инструменты, которые генерируют случайные входные данные и анализируют результаты выполнения программы.

При проведении тестирования черного ящика также используются различные методы оценки покрытия кода тестами, такие как покрытие строк кода, покрытие ветвей и др. Оценка покрытия кода тестами позволяет определить, какие части программы были протестированы, а какие нет.

2.3. Тестирование белого ящика.

Тестирование белого ящика проводится с знанием внутренней структуры программы. Для этого используются различные методы, такие как тестирование модульности, тестирование интеграции, тестирование производительности и др.

Тестирование модульности проверяет корректность работы отдельных модулей программы. Например, если программа состоит из нескольких функций, то тестирование модульности проверит корректность работы каждой функции отдельно.

Тестирование интеграции проверяет корректность работы программы в целом при взаимодействии отдельных модулей. Например, если программа состоит из нескольких функций, то тестирование интеграции проверит корректность работы программы при вызове этих функций в определенном порядке.

Тестирование производительности проверяет скорость работы программы и ее способность обрабатывать большое количество данных. Например,

если программа должна обрабатывать большой объем данных, то тестирование производительности проверит, насколько быстро программа обрабатывает эти данные.

При проведении тестирования белого ящика также используются различные методы оценки покрытия кода тестами, такие как покрытие ветвей, покрытие условий и др. Оценка покрытия кода тестами позволяет определить, какие части программы были протестированы, а какие нет, и помогает выявить ошибки в коде программы.

2.4. Профилирование.

Профилирование - это процесс анализа работы программы с целью выявления узких мест и оптимизации ее работы. Профилирование может проводиться как на этапе разработки программы, так и после ее выпуска.

Профилирование проводится с помощью специальных инструментов - профайлеров. Профайлеры могут быть интегрированы в IDE или работать как отдельные приложения.

Процесс профилирования состоит из нескольких этапов:

1. Подготовка тестовых данных и сценариев работы программы.

2. Запуск профайлера и подключение его к программе.

3. Выполнение тестовых сценариев и сбор информации о работе программы.

4. Анализ полученных данных и выявление узких мест.

5. Оптимизация работы программы на основе полученных результатов.

В процессе профилирования могут использоваться различные методы анализа, такие как анализ времени выполнения функций, анализ использования памяти, анализ вызовов функций и др.

После проведения профилирования может быть создан отчет, в котором будут отражены результаты анализа работы программы и рекомендации по ее оптимизации.

2.5. Мониторинг.

Мониторинг - это процесс наблюдения за работой системы с целью выявления проблем и предотвращения возможных сбоев. Мониторинг может проводиться как на этапе разработки системы, так и после ее запуска.

Мониторинг может проводиться с помощью различных инструментов, таких как мониторы ресурсов, системы журналирования, мониторы производительности и др.

Процесс мониторинга состоит из нескольких этапов:

1. Определение целей мониторинга и выбор инструментов для его проведения. 2. Установка и настройка мониторинговых инструментов. 3. Сбор информации о работе системы, включая данные о производительности, использовании ресурсов и др. 4. Анализ полученных данных и выявление проблем и узких мест. 5. Принятие мер по устранению проблем и оптимизации работы системы.

В процессе мониторинга могут использоваться различные методы анализа, такие как анализ журналов событий, анализ производительности, анализ использования ресурсов и др.

После проведения мониторинга может быть создан отчет, в котором будут отражены результаты анализа работы системы и рекомендации по ее улучшению.

2.6. Дебаггинг.

Дебаггинг - это процесс поиска и устранения ошибок в программном коде. Для проведения дебаггинга могут использоваться различные инструменты, такие как отладчики, логирование и тестирование.

Процесс дебаггинга состоит из нескольких этапов:

1. Выявление проблемы. Этот этап может включать анализ сообщений об ошибках, журналов событий и других данных.

2. Воспроизведение проблемы. Необходимо повторить условия, при которых возникает ошибка, чтобы проанализировать ее причину.

3. Анализ кода. Используя отладчик или другие инструменты, необходимо проанализировать код и выявить места, где возникает ошибка.

4. Исправление ошибки. После выявления места ошибки необходимо внести изменения в код, чтобы исправить проблему.

5. Тестирование. После внесения изменений необходимо провести тестирование, чтобы убедиться, что проблема была устранена и другие части программы не были повреждены.

В процессе дебаггинга может использоваться также методика "разделяй и властвуй когда код разбивается на отдельные части, которые тестируются и отлаживаются по отдельности.

После проведения дебаггинга может быть создан отчет, в котором будут отражены результаты анализа ошибок и рекомендации по их устранению.