**Отчет по самостоятельной работе №5**

**по дисциплине МДК 01.02 “Инструментальные средства разработки программного обеспечения”.**

Выполнил: студент

группы 319

Меркушин Евгений Романович

Дата 17.11.2024

**Цель работы**

Цель данной работы – изучить процесс отладки отдельных модулей программного проекта, выявить типичные ошибки и научиться эффективно использовать инструменты отладки для их устранения.

**Основная структура задания**

# Задание 1: Подготовка среды для отладки

1. **Настройка среды разработки**
   * Для начала выберите подходящий инструмент отладки. Например, если вы работаете с Python, это может быть PyCharm или Visual Studio Code с расширением Debugger for Python. Если используете JavaScript, подойдет Chrome DevTools или VSCode с Node.js Debugger.
   * Установите выбранный инструмент и настройте его для подключения к вашему проекту. В большинстве случаев это сводится к созданию конфигурационного файла запуска отладчика (launch.json для VSCode, например).
2. **Подключение отладчика к проекту**
   * Убедитесь, что ваш отладчик корректно подключается к проекту. Это можно проверить, запустив проект в режиме отладки и установив точки останова (breakpoints). При достижении точки останова выполнение программы должно приостановиться, и вы сможете просматривать значения переменных, стек вызовов и т.д.
3. **Стратегия логирования**
   * Если у вас еще нет системы логирования, настройте ее. В Python это может быть logging, в JavaScript – console.log. Логирование должно быть настроено таким образом, чтобы легко отслеживать ошибки и аномалии в поведении программы.
   * Определите уровни логирования (debug, info, warning, error) и настройте фильтры, чтобы выводить только нужную информацию.

# Задание 2: Отладка логики модуля

1. **Выбор модуля для отладки**
   * Выберите конкретный модуль, который вызывает подозрения на наличие ошибок. Это может быть модуль, связанный с обработкой пользовательского ввода, работой с базой данных или другим важным функционалом.
2. **Пошаговая отладка**
   * Установите точки останова в ключевых местах модуля и начните пошаговое выполнение кода.
   * Просматривая значения переменных на каждом шаге, ищите несоответствия ожидаемым результатам. Обратите внимание на условия ветвлений, циклы и обработку исключений.
3. **Тестирование входных данных и обработка исключений**
   * Проверяйте корректность обработки различных типов входных данных. Особенно обратите внимание на граничные случаи и некорректные данные.
   * Убедитесь, что исключения обрабатываются должным образом, и программа не падает при возникновении ошибок.
4. **Документирование ошибок и предложения по исправлению**
   * Заведите список выявленных ошибок и предложений по их устранению. Описывайте каждую ошибку максимально подробно, указывая место в коде, условия возникновения и предлагаемое решение.

# Задание 3: Интеграционная отладка

1. **Тестирование после интеграции**
   * После завершения отладки отдельного модуля протестируйте его работу в контексте всего проекта. Это позволит выявить потенциальные проблемы взаимодействия с другими модулями.
2. **Использование отладчика для мониторинга работы модуля**
   * Снова используйте отладчик, но теперь установите точки останова в тех местах, где происходит взаимодействие с другими модулями. Следите за тем, как передаются данные и выполняются операции.
3. **Исправление ошибок и обновление документации**
   * Найденные ошибки фиксируйте и документируйте. Обновите документацию по модулю, добавив информацию об изменениях и новых особенностях поведения.

# Задание 4: Отладка производительности и использования памяти

1. **Анализ производительности и использования памяти**
   * Используйте профилировщики для анализа времени выполнения и потребления памяти вашим модулем. Например, в Python это может быть memory\_profiler и line\_profiler.
   * Ищите места, где происходят длительные вычисления или чрезмерное потребление ресурсов.
2. **Выявление участков для оптимизации**
   * Найдите участки кода, которые могут быть оптимизированы. Это могут быть алгоритмы с высокой временной сложностью, избыточные копирования данных или неэффективные структуры данных.
3. **Предложения по оптимизации и реализация**
   * Предложите конкретные улучшения и внесите соответствующие изменения в код. Например, замените сложные алгоритмы на более эффективные, уменьшите количество копий объектов или перейдите на более подходящие структуры данных.
4. **Тестирование после оптимизации**
   * Повторите тестирование модуля, чтобы убедиться, что производительность действительно улучшилась, и новые ошибки не появились.

# Задание 5: Автоматизированное тестирование и отладка

1. **Создание автоматизированных тестов**
   * Разработайте набор автоматизированных тестов для вашего модуля. Это могут быть юнит-тесты, интеграционные тесты или тесты API. В Python часто используют библиотеки unittest или pytest, в JavaScript – Jest или Mocha.
2. **Регрессионное тестирование**
   * Проведите регрессионное тестирование, чтобы убедиться, что изменения, внесенные во время отладки, не нарушают уже существующую функциональность.
3. **Исправление ошибок и повторное тестирование**
   * Исправьте все ошибки, выявленные в ходе автоматизированного тестирования, и снова выполните тесты, чтобы подтвердить стабильность работы модуля.

# Задание 6: Отладка проблем с параллелизмом

1. **Поиск ошибок параллелизма**
   * Если ваш проект использует многопоточность или асинхронное программирование, особое внимание уделите поиску ошибок, связанных с параллельными операциями. Это могут быть гонки данных, дедлоки или другие проблемы синхронизации.
2. **Инструменты и методы для отладки многопоточных приложений**
   * Используйте специальные инструменты для отладки многопоточного кода. Например, в Python это может быть threading или multiprocessing, в JavaScript – async/await и Promise.
   * Применяйте механизмы синхронизации, такие как блокировки (Lock) или семафоры (Semaphore), чтобы избежать гонок данных и других проблем.
3. **Решения для предотвращения ошибок**
   * Предложите и внедрите решения для предотвращения ошибок параллелизма. Это может включать рефакторинг кода, добавление блокировок или использование атомарных операций.

**Вывод**

Из приведённых заданий можно сделать несколько важных выводов:

1. **Отладка — ключевой процесс разработки**: Независимо от сложности проекта, отладка является неотъемлемой частью процесса разработки. Она помогает находить и устранять ошибки, повышать надежность и производительность приложения.
2. **Среда разработки важна**: Правильно настроенная среда разработки и инструменты отладки значительно упрощают процесс поиска и устранения ошибок. Выбор подходящего инструмента зависит от языка программирования и специфики проекта.
3. **Логика и производительность взаимосвязаны**: Проблемы с логикой могут приводить к проблемам с производительностью, поэтому важно тщательно проверять оба аспекта. Оптимизация кода может существенно повысить скорость работы приложения и снизить нагрузку на ресурсы.
4. **Автоматическое тестирование ускоряет процесс**: Использование автоматизированных тестов позволяет быстро выявлять регрессии и гарантировать стабильность работы приложения после внесения изменений. Это особенно полезно при работе над большими проектами.
5. **Параллелизм требует особого внимания**: Многопоточность и асинхронное программирование открывают возможности для повышения производительности, но также создают дополнительные риски, связанные с ошибками синхронизации. Правильная организация параллельных процессов и использование механизмов синхронизации являются ключевыми факторами успеха.