**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

1. Поясните суть тестирования как части верификации программного обеспечения.

|  |
| --- |
| Тестирование ― это практическая часть верификации ПО, направленная на проверку соответствия программы требованиям и ожиданиям. Если **верификация** отвечает на вопрос *«Правильно ли мы создаём продукт?»*, то тестирование даёт фактическое подтверждение через запуск программы, выполнение тестов и поиск дефектов.  Тестирование — это метод верификации, при котором ПО реально выполняется, чтобы выявить ошибки и убедиться, что оно работает правильно. |

2. Изложите основные положения теории отладки и тестирования.

|  |
| --- |
| Основные положения:   1. **Невозможность полного тестирования** – проверить все варианты входных данных и состояний нереально. 2. **Тестирование показывает наличие ошибок, но не их отсутствие**. 3. **Раннее тестирование** снижает стоимость исправлений. 4. **Нельзя полагаться только на тестирование** – нужны и другие методы обеспечения качества. 5. **Отладка ≠ тестирование**: отладка устраняет найденные ошибки, а тестирование их выявляет.   **Теория подчёркивает ограничения тестирования и его роль как средства поиска, но не доказательства отсутствия ошибок.** |

3. Раскройте суть принципов тестирования.

|  |
| --- |
| 1. **Тестирование показывает наличие дефектов, а не их отсутствие**. 2. **Полное тестирование невозможно**, поэтому применяются выборочные проверки. 3. **Раннее тестирование** — чем раньше начато, тем дешевле исправления. 4. **Скопление дефектов** — ошибки часто сосредоточены в немногих модулях. 5. **Парадокс пестицида** — одинаковые тесты со временем теряют эффективность, нужны новые. 6. **Тестирование зависит от контекста** — подход меняется в зависимости от типа ПО. 7. **Заблуждение об отсутствии ошибок** — исправление багов не гарантирует полезность системы, важно соответствие требованиям.\   Принципы помогают строить эффективное тестирование, учитывая его ограничения и практические особенности. |

4. Изложите основные цели и задачи тестирования. Раскройте суть этапов тестирования.

|  |
| --- |
| **Цели тестирования:**   * Выявление и документирование дефектов. * Подтверждение соответствия ПО требованиям и ожиданиям. * Повышение качества и надежности продукта.   **Задачи тестирования:**   * Планирование тестов и подготовка данных. * Выполнение тестов и фиксация результатов. * Анализ найденных дефектов и проверка их исправления. * Оценка готовности ПО к эксплуатации.   **Этапы тестирования:**   1. **Планирование** – определение целей, критериев, объема тестирования. 2. **Проектирование тестов** – создание тест-кейсов и сценариев. 3. **Подготовка тестовой среды** – установка ПО, данных, инструментов. 4. **Исполнение тестов** – запуск тестов и регистрация результатов. 5. **Анализ результатов** – выявление ошибок, оценка качества. 6. **Завершение тестирования** – отчетность, рекомендации по выпуску.   Тестирование системно проверяет качество ПО, выявляет дефекты и оценивает готовность продукта к использованию. |

5. Перечислите и раскройте суть методов ручного тестирования.

|  |
| --- |
| 1. **Позитивное тестирование** – проверка, что ПО работает корректно при правильных данных. 2. **Негативное тестирование** – проверка реакции системы на некорректные или неожиданные данные. 3. **Функциональное тестирование** – проверка соответствия функций требованиям. 4. **Нефункциональное тестирование** – проверка производительности, безопасности, удобства использования. 5. **Тестирование сценариев (use-case)** – проверка последовательностей действий пользователя. 6. **Исследовательское тестирование** – изучение программы без заранее подготовленных тестов для выявления скрытых ошибок.   Ручное тестирование основывается на непосредственном взаимодействии с ПО, проверяет его функции и поведение, помогает находить ошибки, которые автоматизация может пропустить. |

6. Поясните сутькомплексного тестирования.

|  |
| --- |
| **Комплексное тестирование** — это системный подход, при котором проверяются все аспекты работы ПО: функциональные и нефункциональные требования, взаимодействие модулей, устойчивость к ошибкам и нагрузкам.  Суть: сочетание разных видов тестирования (функционального, интеграционного, системного, нагрузочного и др.) для максимально полной проверки качества и надежности продукта перед выпуском.  Комплексное тестирование обеспечивает уверенность, что ПО работает правильно во всех ключевых сценариях и условиях. |

7. Перечислите и поясните сутьметодов тестирования.

|  |
| --- |
| **Методы тестирования:**   1. **Статическое тестирование** – анализ документации и кода без запуска программы (ревью, анализ требований, статический анализ). 2. **Динамическое тестирование** – проверка ПО во время выполнения с различными входными данными. 3. **Белый ящик (White-box)** – тестирование с учётом внутренней структуры кода, проверка всех ветвлений и логики. 4. **Чёрный ящик (Black-box)** – проверка работы системы по требованиям без знания внутренней реализации. 5. **Серый ящик (Gray-box)** – комбинация подходов белого и чёрного ящика, тестировщик частично знает внутреннюю логику.   Методы тестирования помогают выбирать подход в зависимости от целей, доступной информации о системе и типа проверяемого ПО. |

8. Перечислите и разъясните сутьвидов и направлений тестирования.

|  |
| --- |
| **Виды и направления тестирования:**   1. **Функциональное тестирование** – проверка соответствия работы ПО требованиям (что система делает). 2. **Нефункциональное тестирование** – проверка качества работы: производительность, безопасность, удобство, стабильность. 3. **Системное тестирование** – проверка всей системы как единого целого, включая интеграцию модулей. 4. **Интеграционное тестирование** – проверка взаимодействия между модулями и компонентами. 5. **Модульное (unit) тестирование** – проверка отдельных компонентов или функций. 6. **Приёмочное тестирование** – проверка ПО готовности к эксплуатации заказчиком. 7. **Регрессионное тестирование** – проверка, что исправления не нарушили существующий функционал. 8. **Нагрузочное и стресс-тестирование** – проверка работы при больших объёмах данных или высокой нагрузке.   Виды и направления тестирования определяют, **что, как и в каких условиях проверяется**, чтобы убедиться в качестве и надежности ПО. |

9. Изложите понятиетест-кейса и его жизненного цикла.

|  |
| --- |
| **Тест-кейс** — это документ, описывающий конкретный сценарий проверки ПО: входные данные, действия, ожидаемый результат, критерии успешного прохождения.  **Жизненный цикл тест-кейса:**   1. **Проектирование** – создание тест-кейса на основе требований. 2. **Рецензирование и утверждение** – проверка корректности и полноты тест-кейса. 3. **Исполнение** – запуск теста и фиксация фактического результата. 4. **Анализ результатов** – сравнение с ожидаемым результатом, выявление дефектов. 5. **Поддержка и обновление** – корректировка тест-кейсов при изменении требований или ПО.   Тест-кейс систематизирует проверку функций и обеспечивает повторяемость тестирования. |

10. Раскройте суть и изложите понятиятест плана и чек-листа.

|  |
| --- |
| **Тест-план** — это документ, описывающий стратегию тестирования: цели, объём, ресурсы, сроки, методы и критерии успешности. Он помогает организовать процесс тестирования и контролировать его выполнение.  **Чек-лист** — это упрощённый инструмент тестирования, список действий или условий для проверки, где фиксируется факт выполнения каждого пункта. Он удобен для быстрых и повторяемых проверок.  **Тест-план** определяет, *как тестировать*, а **чек-лист** — что конкретно проверять. |

11. Изложите сутьработы с тест-кейсами.

|  |
| --- |
| **Работа с тест-кейсами** заключается в систематическом использовании их для проверки ПО:   1. **Создание** – на основе требований формируются сценарии с входными данными и ожидаемым результатом. 2. **Исполнение** – тестировщик выполняет действия и фиксирует фактический результат. 3. **Анализ** – сравнивается фактический и ожидаемый результат, выявляются дефекты. 4. **Поддержка** – обновление тест-кейсов при изменениях требований или функционала.   Тест-кейсы обеспечивают повторяемость, прозрачность и контроль качества тестирования. |

12. Раскройте суть понятия **«**Баг репорт».

|  |
| --- |
| **Баг репорт** — это документ или запись, фиксирующая обнаруженный дефект в ПО.  Содержит: описание ошибки, шаги воспроизведения, ожидаемый и фактический результат, приоритет и серьёзность.  Баг репорт помогает разработчикам быстро понять проблему и исправить её, а тестировщикам — отслеживать прогресс устранения ошибок. |

13. Изложите сутьсерьезности и приоритетов дефектов.

|  |
| --- |
| **Серьёзность (Severity)** — это степень влияния дефекта на работу системы: от незначительных визуальных ошибок до критических сбоев, нарушающих работу программы.  **Приоритет (Priority)** — это очередность исправления дефекта, определяемая его важностью для бизнеса или пользователя.  **Серьёзность** показывает, *насколько критична ошибка*, а **приоритет** — *в каком порядке её исправлять*. |

14. Разъяснитеструктуру документации тестирования.

|  |
| --- |
| **Структура документации тестирования** включает несколько основных компонентов:   1. **Тест-план** – цели, объем, методы, ресурсы, критерии успеха. 2. **Тест-кейсы** – конкретные сценарии с входными данными и ожидаемым результатом. 3. **Чек-листы** – упрощённые списки действий для проверки функционала. 4. **Баг-репорты** – описание найденных дефектов, шаги воспроизведения, серьёзность и приоритет. 5. **Отчёты о тестировании** – результаты выполнения тестов, статистика, выводы о готовности ПО.   Документация обеспечивает **структурированный контроль тестирования**, прозрачность и повторяемость процессов. |

15. Изложите сутьпроцесса тестирования.

|  |
| --- |
| **Суть процесса тестирования** — это систематическая проверка ПО с целью выявления дефектов и оценки соответствия требованиям.  Основные шаги:   1. **Планирование** – определение целей, объёма, критериев. 2. **Проектирование тестов** – создание тест-кейсов и сценариев. 3. **Подготовка среды** – установка ПО и тестовых данных. 4. **Исполнение тестов** – запуск и фиксация результатов. 5. **Анализ и отчетность** – выявление ошибок, оценка качества и готовности к выпуску.   Тестирование обеспечивает **качественный и контролируемый выпуск ПО**, снижает риски и повышает надёжность продукта. |

16. Раскройте понятие «Регрессивное тестирование».

|  |
| --- |
| **Регрессивное тестирование** — это повторная проверка ПО после исправления ошибок или внесения изменений, чтобы убедиться, что ранее работавший функционал продолжает работать корректно.  Оно предотвращает **непреднамеренные сбои** в уже проверенных частях программы и обеспечивает стабильность системы после обновлений. |

17. Раскройте понятие **«**Тестирование производительности».

|  |
| --- |
| **Тестирование производительности** — это проверка ПО на скорость, устойчивость и эффективность работы при различных нагрузках.  Основные цели:   * Определить время отклика системы. * Выявить узкие места и пределы нагрузки. * Проверить стабильность работы при длительной эксплуатации.   Тестирование производительности помогает **обеспечить быструю, надёжную и устойчивую работу системы** в реальных условиях. |

18. Изложите сутьтестирования на этапе сопровождения.

|  |
| --- |
| **Тестирование на этапе сопровождения** — это проверка ПО после выпуска, когда система уже используется пользователями. Оно направлено на выявление новых дефектов, проверку исправленных ошибок и оценку влияния изменений на существующий функционал.  *К примеру,* обновления модуля оплаты тестировщики запускают регрессионные тесты, чтобы убедиться, что процесс оформления заказа и другие функции магазина работают корректно.  Такое тестирование **поддерживает стабильность и надёжность системы на протяжении всего жизненного цикла**. |

19. Раскройте понятие **«**Нефункциональное тестирование».

|  |
| --- |
| **Нефункциональное тестирование** — это проверка качественных характеристик ПО, не связанных напрямую с его функционалом.  Цель: оценить **производительность, безопасность, удобство, стабильность, совместимость и масштабируемость** системы.  К примеру, проверка времени отклика веб-сайта при одновременном доступе 1000 пользователей или тестирование защиты данных от несанкционированного доступа.  Нефункциональное тестирование гарантирует, что ПО **не только работает, но и работает качественно и надёжно** в реальных условиях. |

20. Раскройте понятие **«**Структурное тестирование».

|  |
| --- |
| **Структурное тестирование** (или тестирование «белого ящика») — это проверка ПО с учётом его внутренней структуры, кода и логики работы.  Цель: убедиться, что **все ветвления, циклы и отдельные модули функционируют корректно** и покрыты тестами.  **К примеру,** проверка всех условий в функции расчёта скидки, чтобы каждый вариант выполнения кода был протестирован.  Структурное тестирование помогает находить ошибки **внутри кода**, которые могут быть незаметны при обычной функциональной проверке. |

21. Раскройте понятие **«**Модульное тестирование».

|  |
| --- |
| **Модульное тестирование** — это проверка отдельных компонентов или модулей ПО изолированно от всей системы.  Цель: убедиться, что **каждый модуль работает корректно и соответствует требованиям**, до интеграции с другими частями программы.  **К примеру,** тестирование функции расчёта налогов в бухгалтерской программе отдельно от всего приложения.  Модульное тестирование позволяет **раннее выявление ошибок** и упрощает их исправление на уровне отдельных компонентов. |

22. Раскройте понятие **«**Функциональное тестирование».

|  |
| --- |
| **Функциональное тестирование** — это проверка ПО на соответствие его функциональных возможностей требованиям и спецификациям.  Цель: убедиться, что **система выполняет заявленные функции корректно для всех предусмотренных сценариев**.  К примеру, проверка регистрации пользователя на сайте: ввод корректных и некорректных данных, отправка формы, получение сообщений об ошибках.  Функциональное тестирование гарантирует, что **программа делает то, что от неё ожидают пользователи и заказчики**. |

23. Перечислите и разъясните суть видов ошибок. Изложите структуру отчета об ошибке.

|  |
| --- |
| **Виды ошибок:**   1. **Синтаксические** – нарушения правил языка программирования (например, пропущена точка с запятой). 2. **Логические** – ошибки в алгоритмах, приводящие к неправильным результатам. 3. **Функциональные** – несоответствие работы ПО требованиям. 4. **Интерфейсные** – ошибки в интерфейсе, неудобство или некорректное отображение элементов. 5. **Производительные** – снижение скорости, неправильная работа под нагрузкой. 6. **Безопасности** – уязвимости, которые могут привести к несанкционированному доступу.   **Структура отчета об ошибке (баг-репорта):**   1. **Название/ID ошибки** – краткое описание. 2. **Шаги воспроизведения** – что нужно сделать, чтобы повторить дефект. 3. **Ожидаемый результат** – как должно работать. 4. **Фактический результат** – что произошло на самом деле. 5. **Приоритет и серьёзность** – очередность исправления и критичность. 6. **Дополнительные сведения** – скриншоты, логи, условия среды.   Отчет об ошибке систематизирует информацию для быстрой диагностики и исправления дефектов. |

24. Перечислите источники и пути выявления требований. Раскройте суть уровней и типов требований.

|  |
| --- |
| **Источники требований:**   * Заказчик и конечные пользователи. * Бизнес-аналитики и техническая документация. * Нормативные и правовые акты. * Результаты анализа существующих систем и конкурентов.   **Пути выявления требований:**   * Интервью и опросы пользователей. * Мозговые штурмы и рабочие группы. * Наблюдение за процессами и анализ бизнес-процессов. * Прототипирование и сценарное моделирование.   **Уровни требований:**   1. **Бизнес-требования** – цели и задачи системы для организации. 2. **Пользовательские требования** – ожидания и функции для конечного пользователя. 3. **Системные/функциональные требования** – детали работы ПО, функции и взаимодействия модулей.   **Типы требований:**   * **Функциональные** – что система должна делать. * **Нефункциональные** – характеристики качества: производительность, безопасность, удобство. * **Ограничения (constraints)** – технические, правовые, ресурсные условия работы системы.   Выявление требований обеспечивает **понятное и полное описание того, что должно быть реализовано**, а уровни и типы помогают систематизировать эту информацию. |

25. Раскройте понятие «Интеграционное тестирование».

|  |
| --- |
| **Интеграционное тестирование** — это проверка взаимодействия между модулями или компонентами системы после их объединения.  Цель: убедиться, что **связанные части ПО корректно обмениваются данными и работают совместно**.  **К примеру,** после объединения модуля корзины и модуля оплаты интернет-магазина проверяется, что товары корректно передаются в платежную систему и заказы оформляются правильно.  Интеграционное тестирование выявляет **ошибки взаимодействия между компонентами**, которые невозможно обнаружить при модульном тестировании. |

26. Пояснитеразницу между валидацией и верификацией.

|  |
| --- |
| **Верификация** — это проверка ПО на соответствие требованиям и спецификациям *в процессе разработки*. Она отвечает на вопрос: «Правильно ли мы создаём продукт?»  **Валидация** — это проверка ПО на соответствие ожиданиям пользователя и его нуждам *на практике, после реализации*. Она отвечает на вопрос: «Делаем ли мы правильный продукт?»  **Верификация** контролирует процесс разработки, **валидация** — результат и полезность продукта для пользователя. |

27. Перечислите и изложите суть моделей разработки ПО.

|  |
| --- |
| **Модели разработки ПО:**   1. **Водопадная (Waterfall)** – последовательная разработка: анализ → проектирование → кодирование → тестирование → внедрение. Простая, но плохо адаптируется к изменениям. 2. **V-образная (V-Model)** – расширение водопада с параллельной проверкой на каждом этапе, упор на тестирование. 3. **Итеративная (Iterative)** – разработка частями, каждая итерация включает проектирование, реализацию и тестирование, позволяет корректировать требования по ходу работы. 4. **Спиральная (Spiral)** – комбинация итеративного подхода и анализа рисков; каждая спираль — планирование, анализ рисков, разработка, тестирование. 5. **Гибкая (Agile)** – быстрая, итеративная разработка с постоянной обратной связью от заказчика и частыми релизами. 6. **DevOps / непрерывная интеграция и доставка** – объединяет разработку и эксплуатацию, постоянное тестирование и развертывание, быстрые обновления.   Модели разработки ПО определяют **структуру, последовательность и контроль процессов**, помогают управлять рисками и качеством. |

28. Раскройте понятие **«**Жизненный цикл тестирования».

|  |
| --- |
| **Жизненный цикл тестирования** — это последовательность этапов, через которые проходит тестирование ПО от планирования до завершения.  Этапы:   1. **Планирование тестирования** – цели, стратегии, ресурсы, критерии успешности. 2. **Анализ требований** – изучение документации для создания тестов. 3. **Проектирование тестов** – разработка тест-кейсов и сценариев. 4. **Подготовка тестовой среды** – установка ПО, данных и инструментов. 5. **Исполнение тестов** – запуск тестов и регистрация результатов. 6. **Анализ и отчетность** – выявление ошибок, оценка качества, рекомендации. 7. **Завершение тестирования** – подведение итогов, хранение документации.   Жизненный цикл тестирования обеспечивает **систематический, повторяемый и контролируемый процесс проверки качества ПО**. |

29. Раскройте понятие **«**Автоматизация тестирования».

|  |
| --- |
| **Автоматизация тестирования** — это использование специальных инструментов и скриптов для выполнения тестов без постоянного участия человека.  Цель: **ускорить проверку, повысить повторяемость и точность тестов**, особенно для регрессионного и повторяющегося тестирования.  **Пример:** автоматический запуск тестов для проверки работы интернет-магазина после каждого обновления кода.  Автоматизация позволяет **сокращать время тестирования и снижать риск человеческой ошибки**. |

30. Поясните определения **«**Ошибки, дефекты, сбои, отказы» и приведите примеры.

|  |
| --- |
| **Ошибки (Mistake / Human Error)** – неправильное действие человека при разработке ПО. *Пример:* программист забыл проверить границы массива.  **Дефекты (Defect / Bug)** – несоответствие кода или документации требованиям, вызванное ошибкой. *Пример:* функция возвращает неверное значение при определённых входных данных.  **Сбои (Failure)** – некорректное поведение программы при выполнении, проявление дефекта. *Пример:* приложение вылетает при вводе спецсимволов в форму.  **Отказы (Fault / Incident)** – ситуация, когда система не выполняет требуемую функцию из-за дефекта. *Пример:* платежная система не проводит оплату при высокой нагрузке.  Ошибки приводят к дефектам, дефекты проявляются как сбои, которые могут вызвать отказы системы. |

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

Дан массив из 20 различных чисел. Найти произведение положительных четных элементов и разность нечетных элементов. Сформулировать требования к программному продукту и выполнить анализ и тестирование программных требований в соответствии со свойствами качественных требований.

* 1. **Сформулировать требования к программному продукту.**

|  |
| --- |
| **Функциональные требования:**   * Программа принимает массив из 20 различных целых чисел. * Находит произведение положительных чётных чисел. * Находит разность нечетных чисел (максимальное – минимальное). * Выводит оба результата на экран.   **Нефункциональные требования:**   * Работа программы должна быть корректной для любых 20 различных целых чисел. * Программа должна работать быстро и занимать минимум памяти. * Код должен быть читаемым.   **Качественные свойства требований:**   * **Полнота**: все операции с массивом описаны. * **Проверяемость**: легко проверить на тестовых данных. * **Понятность**: требования простые и конкретные. * **Согласованность**: нет противоречий между функциями. |

* 1. **Написать программу с помощью С++/С#.**

|  |
| --- |
| using System;  using System.Linq;  class Program  {  static void Main()  {  int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};  int prod = a.Where(x => x > 0 && x % 2 == 0).Aggregate(1, (x, y) => x \* y);  int diff = a.Where(x => x % 2 != 0).Max() - a.Where(x => x % 2 != 0).Min();  Console.WriteLine($"Произведение четных: {prod}, Разность нечетных: {diff}");  }  } |

**3. Выполнить тестирование используя тест-кейсы (не менее 10)**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Тип теста |
| 1 | {1..20} | Произв. четных=24681012141618\*20, Разность нечетных=19-1=18 | Функциональный |
| 2 | {1..20 с отрицательными числами} | Произв. четных=результат по положит. четным, Разность нечетных=макс-мин | Позитивный |
| 3 | Все отрицательные числа | Произв. четных=1, Разность нечетных=0 | Позитивный |
| 4 | Все четные положительные | Произв. четных=произведение всех, Разность нечетных=0 | Граничный |
| 5 | Все нечетные положительные | Произв. четных=1, Разность нечетных=макс-мин | Граничный |
| 6 | Случайные числа | Проверка корректности расчетов | Функциональный |
| 7 | Включены нули | Произв. четных=произведение >0, Разность=макс-мин | Позитивный |
| 8 | Только один четный | Произв. четных=этот элемент, Разность нечетных=макс-мин | Граничный |
| 9 | Только один нечетный | Произв. четных=произведение, Разность нечетных=0 | Граничный |
| 10 | Случайный порядок | Проверка корректности расчетов | Функциональный |

**4. Составить баг-репорт с существующими и возможными багами.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Описание дефекта | Шаги воспроизведения | Ожидаемый результат | Фактический результат | Приоритет | Серьезность |
| 1 | Произведение всех четных = 0 при нуле | Ввести массив с 0 | Произведение >0 | 0 | Средний | Высокая |
| 2 | Разность нечетных = отрицательное число | Ввести массив только с отрицательными нечетными | Разность = max-min | Отрицательная | Средний | Средняя |
| 3 | Ошибка при пустом массиве | Массив пустой | Сообщение об ошибке | Исключение | Высокий | Высокая |

1. **Выполнить тестирование, используя unit-test (2–3 теста)**

|  |
| --- |
| using Xunit;  using System.Linq;  public class ArrayTests  {  [Fact]  public void TestEvenProduct()  {  int[] a = {2,4,6};  int prod = a.Where(x => x>0 && x%2==0).Aggregate(1,(x,y)=>x\*y);  Assert.Equal(48, prod);  }  [Fact]  public void TestOddDifference()  {  int[] a = {1,3,5};  int diff = a.Max()-a.Min();  Assert.Equal(4, diff);  }  } |

**6. Выполнить тестирование, используя аннотацию теста и тестовые примеры (не менее 2).**

|  |
| --- |
| using NUnit.Framework;  using System.Linq;  [TestFixture]  public class ArrayAnnotatedTests  {  [TestCase(new int[]{1,2,3,4}, 8)]  [TestCase(new int[]{-2,4,6}, 24)]  public void TestEvenProductCases(int[] arr, int expected)  {  int prod = arr.Where(x=>x>0 && x%2==0).Aggregate(1,(x,y)=>x\*y);  Assert.AreEqual(expected, prod);  }  [TestCase(new int[]{1,3,5}, 4)]  [TestCase(new int[]{7,9,11}, 4)]  public void TestOddDifferenceCases(int[] arr, int expected)  {  int diff = arr.Where(x=>x%2!=0).Max()-arr.Where(x=>x%2!=0).Min();  Assert.AreEqual(expected, diff);  }  } |

* 1. **Составить чек-лист «Структуризатор».**

|  |
| --- |
| * Массив из 20 различных чисел введён. * Произведение положительных чётных элементов рассчитано. * Разность нечетных элементов рассчитана. * Результаты выведены на экран. * Проверка на пустой массив / нули / отрицательные числа проведена. * Тест-кейсы пройдены (не менее 10). * Unit-тесты выполнены и прошли успешно (2–3). * Аннотированные тесты выполнены (не менее 2). * Баги задокументированы. |