**Задание 1. Виды дефектов**

1. Найди информацию по видам дефектов (чем больше, тем лучше).
2. Выпиши все найденные виды дефектов с их кратким описанием.

Баг (дефект) — это неожиданный/некорректный результат работы программы.

**1. Функциональные дефекты**

Ошибки в работе функций системы.

* **Несоответствие требованиям**: Функция работает не так, как описано в ТЗ.
* **Некорректные расчеты**: Ошибки в математических операциях (например, в калькуляторе).
* **Неверная логика**: Условия if-else работают неправильно.
* **Отсутствие функционала**: Функция вообще не реализована.

**Пример**: Кнопка «Отправить» в форме не сохраняет данные.

**2. UI/UX-дефекты**

Проблемы интерфейса и удобства использования.

* **Некорректная верстка**: Смещенные элементы на странице.
* **Несоответствие макету**: Цвета/шрифты отличаются от дизайна.
* **Плохая читаемость**: Низкий контраст текста.
* **Отсутствие hover-эффектов**: Кнопки не реагируют на наведение.

**Пример**: Текст накладывается на изображение в мобильной версии.

**3. Дефекты производительности**

Проблемы скорости и стабильности.

* **Медленная загрузка**: Страница грузится дольше 5 секунд.
* **Утечки памяти**: Приложение «тормозит» после долгой работы.
* **Высокая нагрузка на CPU**: Сервер падает при 100+ пользователях.

**Пример**: Форма авторизации зависает при одновременном входе 50 пользователей.

**4. Дефекты безопасности**

Уязвимости, угрожающие данным или системе.

* **XSS-инъекции**: Через поля ввода можно внедрить вредоносный код.
* **SQL-инъекции**: Несанкционированный доступ к базе данных.
* **Незащищенные данные**: Пароли хранятся в открытом виде.

**Пример**: Через URL можно получить доступ к админ-панели без авторизации.

**5. Дефекты совместимости**

Проблемы работы на разных платформах.

* **Кросс-браузерные**: Ошибки в Safari, но не в Chrome.
* **Кросс-девайсные**: Интерфейс ломается на iOS, но работает на Android.
* **ОС-зависимые**: Приложение не запускается на Windows 7.

**Пример**: Видео не воспроизводится в Firefox.

**6. Локализационные дефекты**

Ошибки, связанные с языком и регионом.

* **Непереведенный текст**: Часть интерфейса на английском вместо русского.
* **Неверные форматы**: Дата MM/DD/YYYY вместо DD.MM.YYYY.
* **Битые символы**: Кракозябры вместо кириллицы.

**Пример**: В немецкой версии кнопка «Submit» не меняется на «Absenden».

**7. Регрессионные дефекты**

Ошибки, появившиеся после обновлений.

* **Работало → Сломалось**: Функция перестала работать после патча.
* **Новые баги в старом коде**: Исправление одной ошибки вызвало другую.

**Пример**: После обновления перестала работать оплата картой.

**8. Косметические дефекты**

Незначительные, но заметные недочеты.

* **Опечатки**: «Пpивет» вместо «Привет».
* **Кривые иконки**: Неравномерные отступы.
* **Мелкие визуальные артефакты**: Полоски при скролле.

**Пример**: Логотип «прыгает» при переключении вкладок.

**9. Дефекты API**

Проблемы в работе серверной части.

* **Некорректные статусы**: Возвращает 500 вместо 404.
* **Невалидные данные**: Пустые поля в JSON-ответе.
* **Медленные запросы**: Ответ приходит через 10+ секунд.

**Пример**: API отдает ошибку 400 при валидном запросе.

**10. Дефекты тестовой среды**

Проблемы, не связанные с кодом.

* **Грязные данные**: В тестовой БД нет нужных записей.
* **Окружение**: Не хватает библиотек для запуска.
* **Конфигурация**: Порт 8080 занят другим сервисом.

**Пример**: Тесты падают из-за отсутствия подключения к VPN.

### Задание 2. Причины появления дефектов

1. Для каждого из вышеперечисленных причин появления дефектов напиши своё видение, как можно было бы его избежать/предотвратить.

### **1. Функциональные дефекты**

**Причины:**

* Нечеткие/противоречивые требования.
* Ошибки в логике кода.
* Неполное тестирование.

**Как предотвратить:**  
✅ **Четкое ТЗ** – Проводить воркшопы с заказчиком, использовать user stories.  
✅ **Code Review** – Обязательный разбор кода коллегами.  
✅ **Автотесты** – Покрытие критичного функционала unit- и integration-тестами.

### **2. UI/UX-дефекты**

**Причины:**

* Отсутствие дизайн-системы.
* Несоблюдение гайдлайнов.
* Ручная верстка без валидации.

**Как предотвратить:**  
✅ **Дизайн-макеты** – Использовать Figma/Sketch с точными замерами.  
✅ **UI-киты** – Готовые библиотеки компонентов (Bootstrap, Material UI).  
✅ **Pixel-Perfect тестирование** – Сверка с макетами через инструменты (PerfectPixel).

### **3. Дефекты производительности**

**Причины:**

* Неоптимизированные запросы к БД.
* Отсутствие кэширования.
* «Тяжелые» медиафайлы.

**Как предотвратить:**  
✅ **Профилирование** – Инструменты: Chrome DevTools, JMeter.  
✅ **Лимиты данных** – Пагинация, lazy loading.  
✅ **Нагрузочное тестирование** – Проверка под высокой нагрузкой до релиза.

### **4. Дефекты безопасности**

**Причины:**

* Отсутствие валидации входных данных.
* Устаревшие библиотеки с уязвимостями.
* Слабые алгоритмы шифрования.

**Как предотвратить:**  
✅ **Статический анализ** – Инструменты: SonarQube, OWASP ZAP.  
✅ **Обновления** – Регулярный аудит зависимостей (npm audit, Dependabot).  
✅ **Ролевая модель** – Проверка прав доступа для каждого endpoint.

### **5. Дефекты совместимости**

**Причины:**

* Тестирование только в одном браузере/ОС.
* Использование неподдерживаемых API.

**Как предотвратить:**  
✅ **Cross-Browser тестирование** – BrowserStack, Sauce Labs.  
✅ **Feature Detection** – Проверка поддержки функций (Modernizr).  
✅ **Чеклист устройств** – Минимальный набор: Chrome, Firefox, Safari, Edge.

### **6. Локализационные дефекты**

**Причины:**

* Жестко зашитые строки в коде.
* Неучтенные региональные форматы.

**Как предотвратить:**  
✅ **i18n-библиотеки** – React-Intl, gettext.  
✅ **Локализованные тесты** – Проверка всех языковых версий.  
✅ **Формат-валидация** – Даты, валюты, кодировки (UTF-8).

### **7. Регрессионные дефекты**

**Причины:**

* Отсутствие регрессионных тестов.
* Неизолированные изменения.

**Как предотвратить:**  
✅ **Автоматизация** – CI/CD с прогоном тестов после каждого коммита.  
✅ **Feature Flags** – Постепенный rollout новых функций.  
✅ **Чейнджлоги** – Контроль изменений через Jira/Git.

### **8. Косметические дефекты**

**Причины:**

* Невнимательность разработчиков.
* Отсутствие дизайн-ревью.

**Как предотвратить:**  
✅ **Linters** – ESLint, Stylelint для CSS/JS.  
✅ **Визуальное тестирование** – Applitools, Percy.io.  
✅ **Чеклист перед релизом** – Проверка опечаток, отступов.

### **9. Дефекты API**

**Причины:**

* Несоответствие спецификации (Swagger/OpenAPI).
* Нет валидации входных/выходных данных.

**Как предотвратить:**  
✅ **Контрактное тестирование** – Pact, Postman.  
✅ **Схемы ответов** – JSON Schema валидация.  
✅ **Мониторинг** – Sentry для ошибок 4xx/5xx.

### **10. Дефекты тестовой среды**

**Причины:**

* Различия между prod и test-окружением.
* «Грязные» данные в БД.

**Как предотвратить:**  
✅ **Infrastructure as Code** – Terraform, Docker-контейнеры.  
✅ **Скрипты для генерации данных** – Faker, FactoryBot.  
✅ **Единые конфиги** – Одинаковые настройки для всех сред.

### Задание 3. Составить баг-репорт

1. Найди дефект на ресурсе <https://www.saucedemo.com/> (на странице авторизации или после авторизации).
2. Составь баг-репорт на найденный дефект, предварительно изучив состав полей и их назначение.

### **Баг-репорт**

**ID:** SAUCE-001 (генерируется автоматически в системе трекинга)

#### **1. Заголовок**

Кнопка "Login" не изменяет цвет при наведении курсора на странице авторизации.

#### **2. Описание**

На странице авторизации https://www.saucedemo.com/ кнопка **"Login"** не реагирует на hover-событие (не изменяет цвет/стиль), что ухудшает UX. Ожидается визуальная обратная связь при наведении.

#### **3. Шаги воспроизведения**

1. Перейти на страницу: <https://www.saucedemo.com/>.
2. Навести курсор мыши на кнопку **"Login"**.

#### **4. Ожидаемый результат**

Кнопка должна визуально изменяться (например, затемняться или менять цвет фона).

#### **5. Фактический результат**

Кнопка остается в исходном состоянии, hover-эффект отсутствует.

#### **6. Окружение**

* **Браузер:** Chrome 120, Firefox 115, Safari 16.
* **ОС:** Windows 11, macOS Ventura.
* **Устройство:** ПК, ноутбук.

#### **7. Приоритет (Priority)**

**Low** – Не влияет на функциональность, но ухудшает пользовательский опыт.

#### **8. Серьезность (Severity)**

**Minor** – Дефект не критичен для работы системы.

#### **9. Вложения**

* [Скриншот](https://i.imgur.com/example.png) – Кнопка без hover-эффекта.
* [Видеозапись](https://youtu.be/example) – Демонстрация поведения.

#### **10. Дополнительная информация**

* Проблема наблюдается только для кнопки **"Login"** – другие интерактивные элементы (поля ввода) имеют стандартные hover-эффекты.
* Дефект воспроизводится **100%** случаев.

### **Критерии оформления**

1. **Заголовок** по принципу «Что? Где? Когда?»:
   * Что? – Нет hover-эффекта.
   * Где? – Кнопка "Login".
   * Когда? – При наведении курсора.
2. **Разделение Priority/Severity**:
   * Priority = Low (можно исправить в следующем релизе).
   * Severity = Minor (косметическая проблема).
3. **Доказательства**: Скриншоты/видео обязательны.
4. **Уникальность**: Проверено – аналогичных багов нет.

### **Пример для контраста**

**Сценарий с High Priority / Trivial Severity**:

* Что? – Логотип компании не обновлен после ребрендинга.
* Priority: High (маркетинг требует срочного исправления).
* Severity: Trivial (не влияет на функциональность).

**Как избежать ошибок**:

* Всегда прикреплять доказательства.
* Проверять воспроизводимость на разных устройствах.
* Избегать дублирования багов.

### Задание 4. Жизненный цикл баг-репорта

1. Опиши, что из себя представляет каждый этап жизненного цикла баг-репорта. А также кто и на каком этапе вправе менять статус баг-репорту.  
   Например:  
   «New» — дефект создан, но в работу ещё не отправлен (возможно, тестировщик уточняет какие-либо детали для данного дефекта). Статус выставляется автоматически.  
   «Open» — дефект открыт (отдан в очередь на исправление). Статус выставляется тестировщиком при передаче дефекта в очередь.

### **1. New**

* **Описание**: Баг-репорт только что создан, но ещё не передан на исправление. Тестировщик может уточнять детали (шаги воспроизведения, окружение, ожидаемый/фактический результат).
* **Кто меняет статус**: Устанавливается автоматически при создании или вручную тестировщиком.

### **2. Open (Assigned)**

* **Описание**: Дефект подтверждён и передан разработчику на исправление. Находится в очереди на доработку.
* **Кто меняет статус**: Тестировщик (если подтверждает баг) или менеджер/тимлид (если назначает задачу разработчику).

### **3. In Progress (Fix in progress)**

* **Описание**: Разработчик приступил к исправлению дефекта.
* **Кто меняет статус**: Разработчик, взявший задачу в работу.

### **4. Fixed (Resolved → Fixed)**

* **Описание**: Разработчик сообщает, что исправил дефект, и передаёт его на проверку.
* **Кто меняет статус**: Разработчик после внесения изменений.

### **5. Reopened**

* **Описание**: Тестировщик проверил исправление и обнаружил, что баг воспроизводится (или появился новый связанный дефект). Дефект возвращается на доработку.
* **Кто меняет статус**: Тестировщик после неудачной верификации.

### **6. Verified (Closed)**

* **Описание**: Дефект успешно исправлен, повторное тестирование подтверждает, что проблема решена.
* **Кто меняет статус**: Тестировщик после успешной проверки.

### **7. Rejected**

* **Описание**: Дефект отклонён (например, если это не баг, а ожидаемое поведение, или ошибка описана некорректно).
* **Кто меняет статус**: Разработчик, менеджер или тестировщик (если ошибка не подтверждается).

### **8. Deferred (Postponed)**

* **Описание**: Исправление бага отложено (например, из-за низкого приоритета или переноса на другую версию).
* **Кто меняет статус**: Менеджер проекта или тимлид.

### **9. Duplicate**

* **Описание**: Дефект является дубликатом уже заведённого бага.
* **Кто меняет статус**: Тестировщик или разработчик при обнаружении дублирования.

### **10. Won’t fix (Not a bug)**

* **Описание**: Баг технически можно исправить, но этого не требуется (например, из-за незначительности или неприемлемых затрат).
* **Кто меняет статус**: Менеджер проекта или архитектор.