bug_workflow_attributes

Дефекты и ошибки

Прежде всего, стоит разобраться с терминологией. В определениях Error/Mistake/Defect/Bug/Failure/Fault три из них переводятся на русский язык как ошибка. Определения из **ISTQB:**

- Просчет (mistake): См. ошибка;
- Помеха (bug): См. дефект;
- Недочет (fault): См. дефект;
- Ошибка (error): Действие человека, которое приводит к неправильному результату;
- Дефект (defect): Изъян в компоненте или системе, который может привести компонент или систему к невозможности выполнить требуемую функцию, например неверный оператор или определение данных. Дефект, обнаруженный во время выполнения, может привести к отказам компонента или системы;
- Отказ (failure): Отклонение компонента или системы от ожидаемого выполнения, эксплуатации или результата.

Неофициальные же источники показывают более широкую картину:

- Ошибка (Error) возникает из-за просчета (Mistake) в написании кода разработчиком;
- Дефект (Defect) это скрытый недостаток в ПО, возникший из-за ошибки в написании кода;
- Когда дефект (Defect) обнаруживается тестировщиком, он называется багом (Bug);
- Если тестировщики упустили дефект и его нашел пользователь, то это сбой (Failure);

Классификация дефектов

Дефекты можно классифицировать по-разному. Для организации важно следовать единой схеме классификации и применять ее ко всем проектам. Некоторые дефекты можно отнести к нескольким классам или категориям. Из-

за этой проблемы разработчики, тестировщики и сотрудники SQA должны стараться быть максимально последовательными при записи данных о дефектах.

Классы дефектов:

- **Дефекты требований и спецификаций** (Requirements and Specifications Defects): Начало жизненного цикла программного обеспечения важно для обеспечения высокого качества разрабатываемого программного обеспечения. Дефекты, введенные на ранних этапах, очень трудно устранить на более поздних этапах. Поскольку многие документы с требованиями написаны с использованием представления на естественном языке, они могут стать
    - Двусмысленными;
    - Противоречивыми;
    - Непонятными;
    - Избыточными;
    - Неточными.
- Некоторые специфические дефекты требований / спецификаций:
    - Дефекты функционального описания: Общее описание того, что делает продукт и как он должен себя вести (входы / выходы), неверно, двусмысленно и / или неполно;
    - Дефекты функций: описываются как отличительные характеристики программного компонента или системы. Дефекты функций связаны с отсутствием, неправильным, неполным или ненужным описанием функций;
    - Дефекты взаимодействия функций: это происходит из-за неправильного описания того, как функции должны взаимодействовать друг с другом;
    - Дефекты описания интерфейсов: это дефекты, которые возникают в описании взаимодействия целевого программного обеспечения с внешним программным обеспечением, оборудованием и пользователями.
- **Дефекты дизайна**: Дефекты дизайна возникают когда неправильно спроектированы: Системные компоненты, Взаимодействие между компонентами системы, Взаимодействие между компонентами и внешним

программным / аппаратным обеспечением или пользователями. Они включают дефекты в конструкции алгоритмов, управления, логики, элементов данных, описаний интерфейсов модулей и описаний внешнего программного обеспечения / оборудования / пользовательского интерфейса. К дефектам дизайна относятся:

  - Алгоритмические дефекты и дефекты обработки: это происходит, когда этапы обработки в алгоритме, описанном псевдокодом, неверны;
  - Дефекты управления, логики и последовательности: Дефекты управления возникают, когда логический поток в псевдокоде неверен;
  - Дефекты данных: Они связаны с неправильным дизайном структур данных;
  - Дефекты описания интерфейсов модулей: эти дефекты возникают из-за неправильного или непоследовательного использования типов параметров, неправильного количества параметров или неправильного порядка параметров;
  - Дефекты функционального описания: к дефектам этой категории относятся неправильные, отсутствующие или неясные элементы дизайна;
  - Дефекты описания внешних интерфейсов: они возникают из-за неправильных описаний дизайна интерфейсов с компонентами COTS, внешними программными системами, базами данных и аппаратными устройствами.

- **Дефекты кода**: Дефекты кодирования возникают из-за ошибок при реализации кода. Классы дефектов кодирования аналогичны классам дефектов дизайна. Некоторые дефекты кодирования возникают из-за непонимания конструкций языка программирования и недопонимания с разработчиками.
  - Алгоритмические дефекты и дефекты обработки:
    - Непроверенные условия overflow and underflow;
    - Сравнение несоответствующих типов данных;
    - Преобразование одного типа данных в другой;
    - Неправильный порядок арифметических операторов;
    - Неправильное использование или пропуск круглых скобок;

        - Потеря точности (Precision loss);
        - Неправильное использование знаков.
    - Дефекты управления, логики и последовательности: этот тип дефектов включает неправильное выражение операторов case, неправильное повторение циклов и пропущенные пути;
    - Типографические дефекты: в основном это синтаксические ошибки, например неправильное написание имени переменной, которые обычно обнаруживаются компилятором, self-reviews, or peer reviews;
    - Дефекты инициализации: этот тип дефектов возникает, когда операторы инициализации пропущены или неверны. Это может произойти из-за недопонимания или отсутствия связи между программистами или программиста и дизайнера, небрежности или непонимания среды программирования;
    - Дефекты потока данных: дефекты потока данных возникают, когда код не следует необходимым условиям потока данных;
    - Дефекты данных: на это указывает неправильная реализация структур данных;
    - Дефекты интерфейса модуля: возникают из-за использования неправильных или несовместимых типов параметров, неправильного количества параметров или неправильного порядка параметров;
    - Дефекты документации кода: когда документация по коду не описывает, что программа на самом деле делает, либо является неполной или двусмысленной;
    - Внешнее оборудование, дефекты программных интерфейсов: эти дефекты возникают из-за проблем, связанных с Системными вызовами, Ссылками на базы данных, Последовательностью ввода / вывода, Использованием памяти, Использованием ресурсов, Обработкой прерываний и исключений, Обменом данными с оборудованием, Протоколами, Форматами, Интерфейсами с файлами сборки, Временными последовательностями.
- **Дефекты тестирования**: Планы тестирования, тестовые наборы, средства тестирования и процедуры тестирования также могут содержать дефекты. Эти дефекты называются дефектами тестирования. Дефекты в планах тестирования лучше всего обнаруживать с помощью методов review.

- Дефекты тестовой обвязки: Для тестирования программного обеспечения на уровне модулей и интеграции необходимо разработать вспомогательный код. Это называется Test Harness или scaffolding code. Test Harness должен быть тщательно спроектирован, реализован и протестирован, поскольку это рабочий продукт, и этот код можно повторно использовать при разработке новых версий программного обеспечения;

Жизненный цикл дефекта

Жизненный цикл дефекта (Defect Lifecycle) – это последовательность этапов, которые проходит дефект на своём пути с момента его создания до окончательного закрытия. Для простоты восприятия изображается в виде схемы с возможными статусами и действиями, которые приводят к смене этих статусов.

Любой этап работы с дефектом обозначается атрибутом «Статус». Статус дефекта показывает, кто на данный момент работает с дефектом и что следует делать с ним. После того, как работу с багом закончил один из участников проекта, он меняет статус дефекта и «перенаправляет» дефект на того, кто должен продолжить работу.

Новый (New) – отчет о дефекте заводится в баг-трекинговую систему в первый раз.

Назначен (Assigned) – отчет о дефекте назначается на соответствующего разработчика.

Открыт (Open) – разработчик берет отчет о дефекте в работу для анализа и исправления.

Исправлен (Fixed) – разработчик сделал необходимые изменения в коде и проверил эти изменения сам. Отчет о дефекте с этим статусом возвращается обратно тестировщику.

Повторное тестирование в режиме ожидания – после исправления дефекта разработчик предоставил конкретный код для повторного тестирования тестировщиком. Тестирование находится на рассмотрении у тестировщика.

Повторное тестирование (Re-testing) – на этой стадии тестировщик выполняет повторное тестирование измененного кода, который был предоставлен разработчиком, для проверки, исправлен ли дефект или нет.

Проверен (Verified) – если дефект не воспроизводится, тестировщик подтверждает, что этот дефект исправлен.

Переоткрыт (Reopened) – если дефект все же воспроизводится, даже после его исправления разработчиком, тестировщик переоткрывает его и назначает на разработчика. Этот дефект проходит через жизненный цикл дефекта еще раз.

Закрыт (Closed) – если тестировщик уверен, что дефект больше не воспроизводится, то он его закрывает. Этот статус означает, что дефект исправлен, протестирован и одобрен.

Дубликат (Duplicate) – если дефект повторяется дважды или есть два бага, которые являются следствием одной причины, то одному из них присваивается данный статус.

Отклонен (Rejected) – если разработчик считает, что этот дефект не является обоснованным или веским, и дефект не будет рассматриваться для исправления или реализации, он его отклоняет.

Отсрочен (Deferred) – ожидается, что дефект, которому присвоили такой статус, будет исправлен в следующих версиях. Причин для присвоения этого статуса может быть несколько: приоритет дефекта низкий, нехватка времени, данный дефект не повлечет больших сбоев в программном продукте.

Не баг (Not a bug) – этот статус присваивается, если в функционал приложения не будет внесено никаких изменений. Например, если заказчик просит изменить цвет или размер кнопок, или текста – это не дефект, а просто изменения в дизайне приложения.

Давайте рассмотрим **жизненный цикл дефекта более подробно.**

1. **Тестировщик находит дефект.**
2. **Тестировщик оформляет отчет о дефекте в баг-трекинговую систему (статус «Новый» (New)) и назначает на разработчика (статус «Назначен» (Assigned)).**
3. **Разработчик проверяет дефект, воспроизводится он или нет, и присваивает ему один из статусов:**

- **«Дубликат» (Duplicate) – похожий дефект уже есть в баг-трекинговой системе;**
- **«Отклонен» (Rejected) – дефект не является веским;**

- **«Отсрочен» (Deferred) – исправление дефекта можно перенести в следующие версии программного продукта;**
- **«Не баг» (Not a bug) – в функционал программного продукта не будет внесено никаких изменений;**
- **«Открыт» (Open) – разработчик взял дефект в работу;**
- **«Исправлен» (Fixed) – разработчик внес изменения в код и проверил их.**

1. **Тестировщик проводит повторное тестирование дефекта (статус «Повторное тестирование» (Re-testing)).**
2. **Если дефект не воспроизводится, тестировщик закрывает его (статусы «Проверен» (Verified), «Закрыт» (Closed)).**
3. **Если дефект воспроизводится, тестировщик возвращает его разработчику на исправление (статусы «Переоткрыт» (Reopened), «Назначен» (Assigned)) и такой дефект проходит этот жизненный**

Локализация дефекта

Чтобы локализовать баг, необходимо собрать максимальное количество информации о его воспроизведении:

- ***Выявить причины возникновения дефекта***

Например, не проходит восстановление пароля. Необходимо выявить, откуда приходит запрос на сервер в неверном формате — от backend либо frontend.

- ***Проанализировать возможность влияния найденного дефекта на другие области***

Например, в одной из форм, которую редко используют, возникает ошибка при нажатии на кнопку «Редактировать». Если в качестве временного варианта решения проблемы скрыть кнопку, это может повлиять на аналогичную форму в другом окне/вкладке, к которой пользователи обращаются чаще. Для качественного анализа необходимо знать, как работает приложение и какие зависимости могут быть между его частями.

- ***Отклонение от ожидаемого результата***

Нужно проверить, соответствует ли результат тестирования ожидаемому результату. Справиться с этой задачей помогает техническое задание (в

частности, требования к продукту), где должна быть задокументирована информация о тестируемом ПО и его функционировании. Пропускать этот шаг при тестировании не следует: если специалист недостаточно опытен, не зная требований, он может ошибаться и неправильно понимать, что относится к багам, а что нет. Внимательное изучение документации позволяет избежать таких ошибок.

- ***Исследовать окружение***

Необходимо воспроизвести баг в разных операционных системах (iOS, Android, Windows и т.д.) и браузерах (Google Chrome, Mozilla, Internet Explorer и др.). При этом нужно проверить требования к продукту, чтобы выяснить, какие системы должны поддерживаться. Некоторые приложения работают только в определенных ОС или браузерах, поэтому проверять другие варианты не нужно.

- ***Проверить на разных устройствах***

Например, desktop-приложение предназначено для использования на компьютерах, поэтому зачастую нет необходимости тестировать его на мобильных устройствах. Для смартфонов в идеале должна быть разработана отдельная мобильная версия, которую, в свою очередь, нет смысла тестировать на компьютерах. Кроме того, есть web-приложения, которые могут работать и на компьютерах, и на мобильных устройствах, а тестировать их нужно на всех типах устройств. Для тестирования можно использовать эмулятор той или иной среды, но в рамках статьи мы не будем затрагивать этот вопрос.

Мобильную версию приложения нужно тестировать на нескольких устройствах с разной диагональю экрана. При этом можно руководствоваться требованиями к ПО, в которых должно быть указано, с какими устройствами это ПО совместимо.

- ***Проверить в разных версиях ПО***

Для того, чтобы не запутаться в реализованных задачах, в разработке используют версионность ПО. Иногда тот или иной баг воспроизводится в одной версии продукта, но не воспроизводится в другой. Этот атрибут обязательно необходимо указывать в баг-репорте, чтобы программист понимал, в какой ветке нужно искать проблему.

- ***Проанализировать ресурсы системы***

Возможно, дефект был найден при нехватке внутренней или оперативной памяти устройства. В таком случае баг может воспроизводиться на идентичных устройствах по-разному.

Для того, чтобы оптимизировать сроки тестирования, мы рекомендуем использовать техники тест-дизайна.

Фиксирование доказательств

Доказательства воспроизведения бага нужно фиксировать при помощи логов, скринов или записи экрана.

- Логи (лог-файлы или журнал) — это файлы, содержащие системную информацию работы сервера или компьютера, в них хранят информацию об определенных действиях пользователя или программы. Опытному разработчику бывает достаточно посмотреть в логи, чтобы понять, в чем заключается ошибка. Обычно логи прикрепляют к баг-репорту в виде .txt-файла.

Live-логирование – это снятие системных логов в режиме реального времени. Для этого можно использовать следующие программы: Fiddler, Visual Studio для Windows, iTools, Xcode для iOS, Android Debug Monitor, Android Studio для Android и др.

- Скрин (снимок) экрана. При ошибках в интерфейсе снимок помогает быстрее разобраться в проблеме. Программ для снятия скриншотов очень много, каждый QA-специалист может использовать те, которые ему наиболее удобны: Jing, ShareX, Lightshot и др.
- Скринкаст (англ. screen – экран, broadcasting – трансляция) – это запись видеоизображения с экрана компьютера или другого цифрового устройства. Это один из самых эффективных способов поделиться тем, что происходит на экране монитора. Таким способом QA-специалисту легко наглядно показать ошибки в работе любого программного продукта. Сделать запись с экрана помогут незаменимые инструменты любого QA-специалиста: Snagit, Monosnap, Movavi Screen Capture, Jing, Reflector, ADB Shell Screenrecord, AZ Screen Recorder и др.
- Рекордер действий. Программные средства дают возможность воспроизвести все записанные движения мышки и действия, производимые на клавиатуре компьютера. Примеры программ – Advanced Key and Mouse Recorder для desktop-платформ.

Оформление баг-репорта

Все найденные дефекты обязательно нужно документировать, чтобы каждый задействованный на проекте специалист мог получить инструкции по воспроизведению обнаруженного дефекта и понять, насколько это критично. Если в команде принято устно передавать разработчику информацию о найденных дефектах, есть риск упустить что-то из вида.

Дефект, который не задокументирован – не найден!

Когда вся необходимая информация собрана, а баг локализован, можно приступать к оформлению баг-репорта в таск-трекере. Чем точнее описание бага, тем меньше времени нужно для его исправления. Список атрибутов для каждого проекта индивидуален, но некоторые из них – например, шаги воспроизведения, ожидаемый результат, фактический результат – присутствуют практически всегда.

Атрибуты баг-репорта:

1. Название

Баг должен быть описан кратко и ёмко, иметь понятное название. Это поможет разработчику разобраться в сути ошибки и в том, может ли он взять этот случай в работу, если занимается соответствующим разделом системы. Также это позволяет упростить подключение новых специалистов на проект, особенно если разработка ведется много лет подряд, а запоминать баги и отслеживать их в таск-трекере становится все сложнее. Название проекта можно составлять по принципу «Где? Что? Когда?» или «Что? Где? Когда?», в зависимости от внутренних правил команды.

Например:

Где происходит? — В карточке клиента (в каком разделе системы).

Что именно происходит? — Не сохраняются данные.

Когда происходит? — При сохранении изменений.

2. Проект (название проекта)

3. Компонент приложения

В какой части функциональности тестируемого продукта найден баг.

4. Номер версии

Версия продукта, ветка разработки, в которой воспроизводится ошибка.

5. Критичность

Этот атрибут показывает влияние дефекта на функциональность системы, например:

- Blocker — дефект, блокирующий использование системы.
- Critical — ошибка, которая нарушает основную бизнес-логику работы системы.
- Major — ошибка, которая нарушает работу определенной функции, но не всей системы.
- Minor — незначительная ошибка, не нарушающая бизнес-логику приложения, например, ошибка пользовательского интерфейса.
- Trivial — малозаметная, неочевидная ошибка. Это может быть опечатка, неправильная иконка и т.п.

6. Приоритет

Приоритет определяет, насколько срочно нужно исправить ошибку. Обычно выделяют следующие виды приоритетов:

High — ошибка должна быть исправлена как можно скорее, является критичной для проекта.

Medium — ошибка должна быть исправлена, но её наличие не является критичным для проекта.

Low — ошибка должна быть исправлена, но не требует срочного решения.

7. Статус

8. Автор баг-репорта

9. На кого назначен

Баг-репорт отправляют тимлиду проекта или разработчику, который будет заниматься исправлением дефекта, в зависимости от принятых в команде договоренностей.

10. Окружение

Где найден баг: операционная система, наименование и версия браузера.

11. Предусловие (если необходимо)

Необходимо для описания действий, которые предшествовали воспроизведению бага. Например, клиент авторизован в системе, создана заявка с параметрами ABC и т.д. Баг-репорт может не содержать предусловие, но иногда оно бывает необходимо для того, чтобы проще описать шаги воспроизведения.

12. Шаги воспроизведения

Один из самых важных атрибутов — описание шагов, которые привели к нахождению бага. Оптимально использовать 5-7 понятных и кратких шагов для описания бага, например:

1. Открыть (...)
2. Кликнуть (…)
3. Ввести в поле A значение N1
4. Ввести в поле B значение N2
5. Кликнуть кнопку «Calculate»

13. Фактический результат

Что произошло после воспроизведения указанных выше шагов.

14. Ожидаемый результат

Что должно произойти после воспроизведения шагов тестирования, согласно требованиям.

15. Прикрепленный файл

Логи, скриншоты, видеозапись экрана — всё, что поможет разработчику понять суть ошибки и исправить ее.

После составления баг-репорта обязательно нужно проверить его, чтобы избежать ошибок или опечаток.

Локализация и оформление багов — необходимые составляющие работы QA-специалиста с программным продуктом.