Написание баг репорта

Баг репорт - это технический документ и в связи с этим хотим отметить, что язык описания проблемы должен быть техническим. Должна использоваться правильная терминология при использовании названий элементов пользовательского интерфейса (editbox, listbox, combobox, link, text area, button, menu, popup menu, title bar, system tray и т.д.), действий пользователя (click link, press the button, select menu item и т.д.) и полученных результатах (window is opened, error message is displayed, system crashed и т.д.).

Требования к обязательным полям баг репорта

Отметим, что обязательными полями баг репорта являются: **короткое описание** (*Bug Summary*), **серьезность** (*Severity*), **шаги к воспроизведению** (*Steps to reproduce*), **результат** (*Actual Result*), **ожидаемый результат** (*Expected Result*). Ниже приведены требования и примеры по заполнению этих полей.

Короткое описание

Название говорит само за себя. В одном предложение вам надо уместить смысл всего баг репорта, а именно: коротко и ясно, используя правильную терминологию сказать что и где не работает. Например:

1. Приложение зависает, при попытке сохранения текстового файла размером больше 50Мб.
2. Данные на форме "Профайл" не сохраняются после нажатия кнопки "Сохранить".

В дополнение предлагаем вам изучить Принцип "Где? Что? Когда?".

> *"В чем этот принцип состоит?*
> *Составьте предложение, в котором факты дефекта изложены в следующей последовательности:*
>
> - ***Где?**: В каком месте интерфейса пользователя или архитектуры программного продукта находится проблема. Причем, начинайте предложение с существительного, а не предлога.*
> - ***Что?**: Что происходит или не происходит согласно спецификации или вашему представлению о нормальной работе программного продукта. При этом указывайте на наличие или отсутствие объекта проблемы, а не на его содержание (его указывают в описании). Если содержание проблемы варьируется, все известные варианты указываются в описании.*
> - ***Когда?**: В какой момент работы программного продукта, по наступлению какого события или при каких условиях проблема проявляется.*
>
> *Почему последовательность должна быть именно такой?*
> *В таком виде незнакомые дефекты удобнее сортировать по summary как показывает практика (ведь, скорее всего, именно среди дефектов других инженеров будет производиться поиск дубликатов). Если вы другого мнения - придумайте свою последовательность, но она должна стать*

единой для всех без исключения членов проекта, иначе вы не добьетесь необходимого результата."

Серьезность

В двух словах можно отметить, что если проблема найдена в ключевой функциональности приложения и после ее возникновения приложение становится полностью недоступно, и дальнейшая работа с ним невозможна, то она блокирующая. Обычно все блокирующие проблемы находятся во время первичной проверки новой версии продукта, т.к. их наличие не позволяет полноценно проводить тестирование. Если же тестирование может быть продолжено, то серьезность данного дефекта будет критическая. На счет значительных, незначительных и тривиальных ошибок вопрос достаточно прозрачный и на наш взгляд не требует лишних объяснений.

Шаги к воспроизведению / Результат / Ожидаемый результат

Очень важно четко описать все шаги, с упоминаем всех вводимых данных (имени пользователя, данных для заполнения формы) и промежуточных результатов.

Например:

Шаги к воспроизведению
1. Войдите в системы: Пользователь Тестер1, пароль xxxXXX
--> Вход в систему осуществлен
2. Кликните линк Профайл
--> Страница Профайл открылась
3. Введите Новое имя пользователя: Тестер2
4. Нажмите кнопку Сохранить
Результат
На экране появилась ошибка. Новое имя пользователя не было сохранено
Ожидаемый результат
Страница профайл перегрузилась. Новое значение имени пользователя сохранено.

Основные ошибки при написании багов репортов

Недостаточность предоставленных данных
Не всегда одна и таже проблема проявляется при всех вводимых значениях и под любым вошедшим в систему пользователем, поэтому настоятельно рекомендуется вносить все необходимые данные в баг репорт

Определение серьезности
Очень часто происходит либо завышение, либо занижение серьезности дефекта, что может привести к неправильной очередности при решении проблемы.

Язык описания
Часто при описании проблемы используются неправильная терминология или сложные речевые обороты, которые могут ввести в заблуждение человека, ответственного за решение проблемы.

Отсутствие ожидаемого результата
В случаях, если вы не указали, что же должно быть требуемым поведением системы, вы тратите время разработчика, на поиск данной информации, тем самым замедляете исправления дефекта. Вы должны указать пункт в требованиях, написанный тест кейс или же ваше личное мнение, если эта ситуация не была документирована.

Заполнение полей баг репорта

В описанной ниже таблице представлены основные поля баг репорта и роль работника, ответственного за заполнение данного поля. Красным цветом выделены обязательные для заполнения поля:

Поле	Ответственный за заполнение поля
Короткое описание (Summary)	Автор баг репорта (обычно это Тестировщик)
Проект (Project)	Автор баг репорта (обычно это Тестировщик)
Компонент приложения (Component)	Автор баг репорта (обычно это Тестировщик)
Номер версии (Version)	Автор баг репорта (обычно это Тестировщик)
Серьезность (Severity)	Автор баг репорта (обычно это Тестировщик), однако данный атрибут может быть изменен вышестоящим менеджером
Приоритет (Priority)	Менеджер проекта или менеджер ответственный за разработку компонента, на который написан баг репорт
Статус (Status)	Автор баг репорта (обычно это Тестировщик), но многие системы баг трекинга выставляют статус по умолчанию
Автор (Author)	Устанавливается по умолчанию, если нет, то указывается имя автора баг репорта
Назначен на (Assigned To)	Менеджер проекта или менеджер ответственный за разработку компонента, на который написан баг репорт
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Автор баг репорта (обычно это Тестировщик)
Шаги воспроизведения (Steps to Reproduce)	Автор баг репорта (обычно это Тестировщик)
Фактический Результат (Result)	Автор баг репорта (обычно это Тестировщик)
Ожидаемый результат (Expected Result)	Автор баг репорта (обычно это Тестировщик)
Прикрепленный файл (Attachment)	Автор баг репорта (обычно это Тестировщик), а также любой член командной группы, считающий, что прикрепленные данные помогут в исправлении бага

Принцип "Где? Что? Когда?"

В ходе работы с дефектами часто сталкиваешься с тем, что приходится обращать внимание инженеров на необходимость более четкой формулировки summary дефекта. Причем, со временем это повторяется снова и снова. Та же ситуация наблюдается на интервью с соискателями на должность инженера по тестированию. Многие компании уделяют этому

недостаточно внимания, но в сколь-нибудь большом проекте это может стать проблемой требующей решения.

Так почему же мы снова и снова пишем неправильные summary?
Во первых, нет четко сформулированного правила построения summary. Все знают что оно должно быть коротким и понятным и настроить человека, читающего его на правильное восприятие собственно описания самого дефекта. Многие соглашаются с тем, что оно должно быть уникальным, чтобы облегчить поиск дубликатов, которые в свою очередь, есть признак потерь продуктивности работы команды, но при этом все задаются вопросом, а какова тогда должна быть степень уникальности. В общем случае дубликаты могут рождаться по трем основным причинам:

1. Summary дефектов в большинстве своем нечетки. Это приводит к тому, что при поиске дубликатов своего дефекта инженер не может распознать дефект по summary и вынужден часто читать описание каждого подобного дефекта, что серьезно увеличивает время создания каждого отдельного дефекта в базе. Поэтому, скорее всего, он не будет перечитывать описание тщательно, чтобы не снижать свою продуктивность.
2. Инженеры еще недостаточно хорошо освоили сам продукт и связанную с ним предметную область, поэтому создают дефекты с различными неполными описаниями, но имеющими один и тот же источник.
3. Неправильное планирование тестирования (например когда области тестирования 2-х независимых инженеров сильно перекрываются, а поиск дубликатов к тому же затруднен причиной № 1).

В общем случае дубликаты - достаточно серьезная проблема, приводящая к снижению продуктивности как команды тестирования, так и программистов. Поэтому каждую причину надо устранять в отдельности. Способы устранения причин № 2 и № 3 достаточно очевидны. Причина № 2 устраняется дополнительными тренингами с формальным отведением времени под них при планировании либо естественным путем - инженеры со временем разберутся в продукте на достаточном уровне сами (метод решения зависит от степени влияния причины на проблему). Причина № 3 устраняется путем более четкого планирования тестирования с явным указанием областей ответственности инженеров в точках интеграции смежных компонентов). А вот причина № 1 системная, и ее устранение состоит в систематическом усвоении норм построения summary, которые должны стать частью вашего процесса и при этом не усложнять его, а упрощать.

Во вторых, если правила построения summary сложны и нечетки - они так или иначе в рутинном потоке заданий инженерами частично или полностью забываются.

Есть разные способы построения summary на сегодня. Вот, к примеру, два из них.

Первый способ:

1. Напишите сначала описание дефекта.
2. Посмотрите на него внимательно и выделите из него ключевые моменты.
3. Попробуйте сложить эти ключевые моменты вместе.

То что у вас получится в итоге и будет составлять summary вашего дефекта.

Второй способ:

Напишите описание дефекта по следующей схеме:

1. Шаги воспроизведения
2. Ожидаемый результат
3. Полученный результат

Если Вы построили описание правильно, то "Полученный результат" и будет summary вашего дефекта в общем виде. При необходимости можно добавить в него недостающие детали.

Я предлагаю другой способ или так называемый **принцип "Где? Что? Когда?"**. Именно в таком порядке, не путайте с названием популярной телепередачи :)

В чем этот принцип состоит?
Составьте предложение, в котором факты дефекта изложены в следующей последовательности:

1. **Где?**: В каком месте интерфейса пользователя или архитектуры программного продукта находится проблема. Причем, начинайте предложение с существительного, а не предлога.
2. **Что?**: Что происходит или не происходит согласно спецификации или вашему представлению о нормальной работе программного продукта. При этом указывайте на наличие или отсутствие объекта проблемы, а не на его содержание (его указывают в

описании). Если содержание проблемы варьируется, все известные варианты указываются в описании.
3. **Когда?**: В какой момент работы программного продукта, по наступлению какого события или при каких условиях проблема проявляется.

Почему последовательность должна быть именно такой?
В таком виде незнакомые дефекты удобнее сортировать по summary как показывает практика (ведь, скорее всего, именно среди дефектов других инженеров будет производиться поиск дубликатов). Если вы другого мнения - придумайте свою последовательность, но она должна стать единой для всех без исключения членов проекта, иначе вы не добьетесь необходимого результата.

Вот пример такого summary.
Допустим, у Вас есть диалог "Преобразовать данные" с кнопкой "Преобразовать".
И при нажатии этой кнопки появляется сообщение об ошибке "Ошибка №...".
Тогда summary дефекта будет состоять из следующих частей:
1. **Где?**: Диалог "Преобразовать данные"
2. **Что?**: показывается сообщение об ошибке
3. **Когда?**: при нажатии кнопки "Преобразовать"

Итоговое summary будет иметь следующий вид:

Диалог "Преобразовать данные" показывает сообщение об ошибке при нажатии кнопки "Преобразовать"

Все остальные детали должны быть указаны в описании дефекта.

Основные поля баг / дефект репорта

Разные системы менеджмента дефектами, предлагают нам разные поля для заполнения и разные структуры описания дефектов. Нижеприведенная таблица - это попытка показать то, что на основании полученного нами опыта, мы рекомендуем вам использовать в виде **шаблона баг репорта**.

Шапка	
Короткое описание (Summary)	Короткое описание проблемы, явно указывающее на причину и тип ошибочной ситуации.
Проект (Project)	Название тестируемого проекта
Компонент приложения (Component)	Название части или функции тестируемого продукта
Номер версии (Version)	Версия на которой была найдена ошибка
Серьезность (Severity)	Наиболее распространена пятиуровневая система градации серьезности дефекта: • S1 Блокирующий (Blocker) • S2 Критический (Critical) • S3 Значительный (Major) • S4 Незначительный (Minor) • S5 Тривиальный (Trivial) (подробнее смотрите ниже в разделе **Серьезность дефекта)**
Приоритет (Priority)	Приоритет дефекта: • P1 Высокий (High) • P2 Средний (Medium) • P3 Низкий (Low) (подробнее смотрите ниже в разделе **Приоритет дефекта)**
Статус (Status)	Статус бага. Зависит от используемой процедуры и **жизненного цикла бага** (bug workflow and life cycle)
Автор (Author)	Создатель баг репорта
Назначен на (Assigned To)	Имя сотрудника, назначенного на решение проблемы
Окружение	
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Информация об окружении, на котором был найден баг: операционная система, сервис пак, для WEB тестирования - имя и версия браузера и т.д.
...	
Описание	

Шаги воспроизведения (Steps to Reproduce)	Шаги, по которым можно легко воспроизвести ситуацию, приведшую к ошибке.
Фактический Результат (Result)	Результат, полученный после прохождения шагов к воспроизведению
Ожидаемый результат (Expected Result)	Ожидаемый правильный результат
Дополнения	
Прикрепленный файл (Attachment)	Файл с логами, скриншот или любой другой документ, который может помочь прояснить причину ошибки или указать на способ решения проблемы

Серьезность и Приоритет Дефекта

Разные системы баг трекинга предлагают нам разные пути описания серьезности и приоритета баг репорта, неизменным остается лишь смысл, вкладываемый эти поля.
Серьезность (**Severity**) - это атрибут, характеризующий влияние дефекта на работоспособность приложения.
Приоритет (**Priority**) - это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.

Градация Серьезности дефекта (Severity)
S1 Блокирующая (Blocker)
Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.
S2 Критическая (Critical)
Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.
S3 Значительная (Major)
Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.
S4 Незначительная (Minor)
Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.
S5 Тривиальная (Trivial)
Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Градация Приоритета дефекта (Priority)
P1 Высокий (High)
Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.
P2 Средний (Medium)
Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.
P3 Низкий (Low)
Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.
Порядок исправления ошибок по их приоритетам:
High -> Medium -> Low

Требования к количеству открытых багов

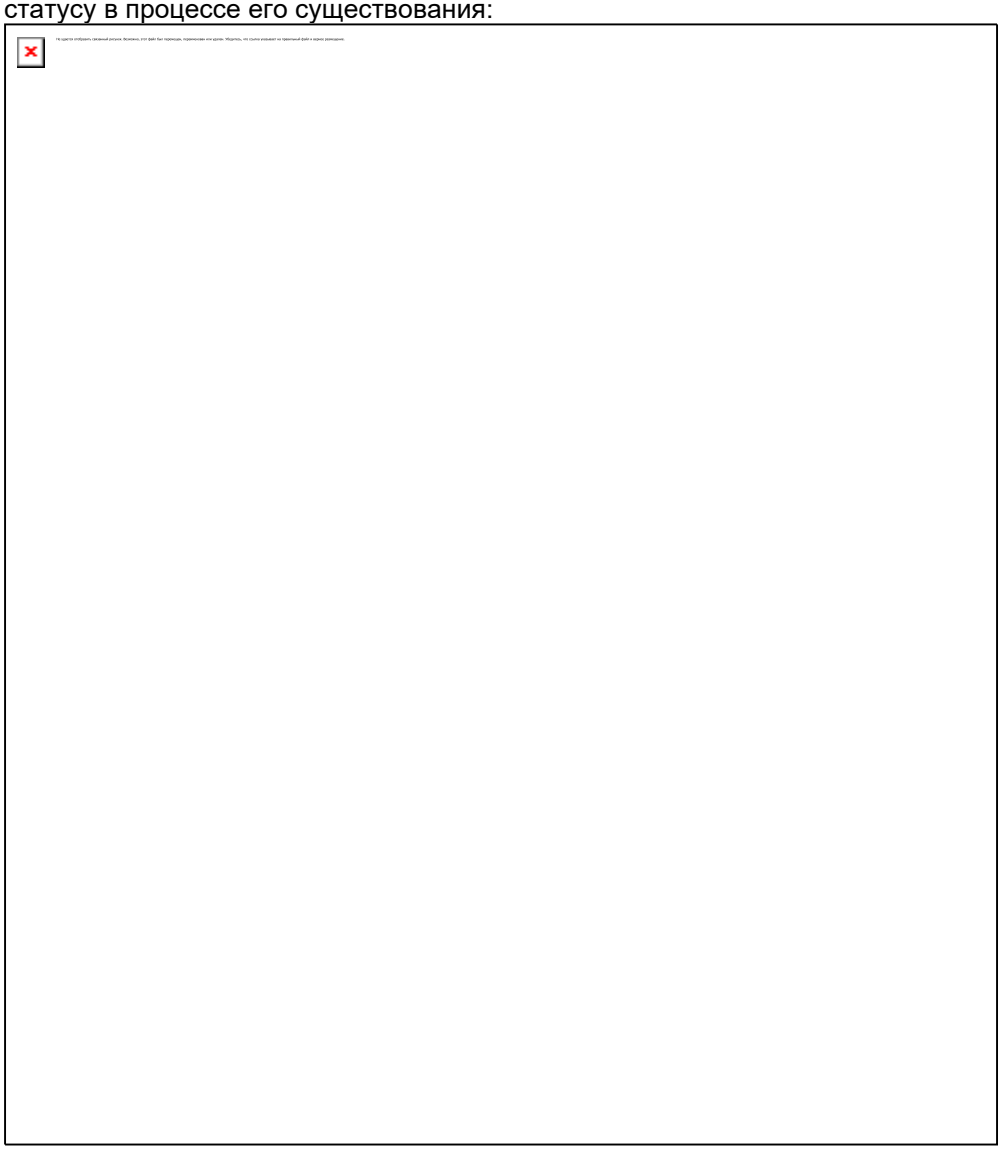
Хотим предложить вам следующий подход к определению требований к количеству открытых багов:

- Наличие открытых дефектов P1, P2 и S1, S2, считается неприемлемым для проекта. Все подобные ситуации требуют срочного решения и идут под контроль к менеджерам проекта.
- Наличие строго ограниченного количества открытых ошибок P3 и S3, S4, S5 не является критичным для проекта и допускается в выдаваемом приложении. Количество же открытых ошибок зависит от размера проекта и установленных критериев качества.

Все требования к открытым ошибкам оговариваются и документируются на этапе принятия решения о качестве разрабатываемого продукта. Как пример документирования подобных требований - это пункт **Критерии окончания тестирования** в плане тестирования.

Жизненный цикл бага

Перед началом описания элементарного жизненного цикла бага предлагаем рассмотреть следующую блок-схему, показывающую основные статусы и возможные переходы от статуса к статусу в процессе его существования:



Теперь переходим к описанию данной схемы.
Допустим вы нашли баг и зарегистрировали его в баг трекинг системе. Согласно нашей блок-схеме он получит статус “Новый”. Тестировщик, ответственный за валидацию новых баг репортов, или координатор проекта (в зависимости от распределения ролей в вашей команде) может перевести его в один из следующих статусов:

- “Отклонен”, если данный баг невалидный или повторный, или же его просто не смогли воспроизвести

- “Отсрочен”, если данный баг не нужно исправлять в данной итерации
- “Открыт”, если исправление бага необходимо

Рассмотрим теперь по порядку каждый из вариантов.

1. **Отклонен**. В этом случае вы можете либо поспорить о судьбе вашего багрепорта, изменив статус на “Переоткрыт” либо закрыть его - статус “Закрыт”
2. **Отсрочен**. Баг репорт в статусе “Отсрочен” можно перевести в статус “Открыт”, когда потребуется исправление либо в статус “Закрыт”, если уже не потребуется.
3. **Открыт**. Именно в таком состоянии разработчик получает баг репорт для исправления. Он может отклонить (дальнейшие действия смотрите в пункте 1) или исправить баг. Баг репорт в статусе “Исправлен” переводится на тестировщика для проверки. В случае если проблема все еще воспроизводится, выставляется статус “Переоткрыт” и баг репорт направляется назад на доработку к разработчику. Если же исправление было успешным, то баг репорт переводится в статус “Закрыт”.

* * *

Хотим отметить, что данная схема сильно упрощена. Для большей наглядности и, возможно, удобства работы на проекте, вы можете добавить дополнительные статусы и переходы, тем более, что современные баг трекинговые системы позволяют это делать. Правда имейте ввиду, что излишне запутанные схемы переходов и лишние статусы могут значительно усложнить жизнь.

Примечание 1: в некоторых системах баг трекинга созданный баг репорт сразу получает статус “Открыт” без дополнительной валидации

Примечание 2: многие баг трекинговые системы позволяют переоткрывать закрытые баги, однако лично я против такой практики, поэтому и не описывал подобный переход в выше представленном жизненном цикле

Примечание 3: Рассмотренный выше жизненный цикл основан на том, что в команде есть кто-то, ответственный за назначение баг репортов. В случае, если такой роли на проекте нет, то баги назначаются разработчиками самостоятельно, и тогда во избежании путанницы, есть смысл ввести еще один промежуточный статус “В разработке” (In progress), показывающий, что данный баг репорт уже назначен и находится на стадии исправления.