2.5.4. Инструментальные средства управления отчётами о дефектах

Так называемые «инструментальные средства управления отчётами о дефектах» в обычной разговорной речи называют «баг-трекинговыми системами», «баг-трекерами» и т.д. Но мы здесь по традиции будем придерживаться более строгой терминологии.

Инструментальных средств управления отчётами о дефектах (bug tracking system, defect management tool[319]) очень много[320], к тому же многие компании разрабатывают свои внутренние средства решения этой задачи. Зачастую такие инструментальные средства являются частями инструментальных средств управления тестированием{125}.

Как и в случае с инструментальными средствами управления тестированием, здесь не имеет смысла заучивать, как работать с отчётами о дефектах в том или ином средстве. Мы лишь рассмотрим общий набор функций, как правило, реализуемых такими средствами:

- Создание отчётов о дефектах, управление их жизненным циклом с учётом контроля версий, прав доступа и разрешённых переходов из состояния в состояние.
- Сбор, анализ и предоставление статистики в удобной для восприятия человеком форме.
- Рассылка уведомлений, напоминаний и иных артефактов соответствующим сотрудникам.
- Организация взаимосвязей между отчётами о дефектах, тест-кейсами, требованиями и анализ таких связей с возможностью формирования рекомендаций.
- Подготовка информации для включения в отчёт о результатах тестирования.
- Интеграция с системами управления проектами.

Иными словами, хорошее инструментальное средство управления жизненным циклом отчётов о дефектах не только избавляет человека от необходимости внимательно выполнять большое количество рутинных операций, но и предоставляет дополнительные возможности, облегчающие работу тестировщика и делающие её более эффективной.

Для общего развития и лучшего закрепления темы об оформлении отчётов о дефектах мы сейчас рассмотрим несколько картинок с формами из разных инструментальных средств.

Здесь вполне сознательно не приведено никакого сравнения или подробного описания — подобных обзоров достаточно в Интернете, и они стремительно устаревают по мере выхода новых версий обозреваемых продуктов.

Но интерес представляют отдельные особенности интерфейса, на которые мы обратим внимание в каждом из примеров (важно: если вас интересует подробное описание каждого поля, связанных с ним процессов и т.д., обратитесь к официальной документации — здесь будут лишь самые краткие пояснения).

---

[319] **Defect management tool, Incident management tool.** A tool that facilitates the recording and status tracking of defects and changes. They often have workflow-oriented facilities to track and control the allocation, correction and re-testing of defects and provide reporting facilities. See also incident management tool. [ISTQB Glossary]

[320] «Comparison of issue-tracking systems», Wikipedia [http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems]

Jira[321]

1. Project (проект) позволяет указать, к какому проекту относится дефект.
2. Issue type (тип записи/артефакта) позволяет указать, что именно представляет собой создаваемый артефакт. JIRA позволяет создавать не только отчёты о дефектах, но и множество других артефактов[322], типы которых можно настраивать[323]. По умолчанию представлены:
   - Improvement (предложение по улучшению) — было описано подробно в разделе, посвящённом полям отчёта о дефекте (см. описание поля «симптом», значение «предложение по улучшению»{176}).
   - New feature (новая особенность) — описание новой функциональности, нового свойства, новой особенности продукта.
   - Task (задание) — некое задание для выполнения тем или иным участником проектной команды.
   - Custom issue (произвольный артефакт) — как правило, это значение при настройке JIRA удаляют, заменяя своими вариантами, или переименовывают в Issue.
3. Summary (краткое описание) позволяет указать краткое описание дефекта.
4. Priority (срочность) позволяет указать срочность исправления дефекта. По умолчанию JIRA предлагает следующие варианты:
   - Highest (самая высокая срочность).
   - High (высокая срочность).
   - Medium (обычная срочность).
   - Low (низкая срочность).
   - Lowest (самая низкая срочность).

   Обратите внимание: по умолчанию поля severity (важность) нет. Но его можно добавить.
5. Components (компоненты) содержит перечень компонентов приложения, затронутых дефектом (хотя иногда здесь перечисляют симптомы дефектов).
6. Affected versions (затронутые версии) содержит перечень версий продукта, в которых проявляется дефект.
7. Environment (окружение) содержит описание аппаратной и программной конфигурации, в которой проявляется дефект.
8. Description (подробное описание) позволяет указать подробное описание дефекта.
9. Original estimate (начальная оценка времени исправления) позволяет указать начальную оценку того, сколько времени займёт устранение дефекта.
10. Remaining estimate (расчётное остаточное время исправления) показывает, сколько времени осталось от начальной оценки.
11. Story points (оценочные единицы) позволяет указать сложность дефекта (или иного артефакта) в специальных оценочных единицах, принятых в гибких методологиях управления проектами.
12. Labels (метки) содержит метки (теги, ключевые слова), по которым можно группировать и классифицировать дефекты и иные артефакты.
13. Epic/Theme (история/область) содержит перечень высокоуровневых меток, описывающих относящиеся к дефекту крупные области требований, крупные модули приложения, крупные части предметной области, объёмные пользовательские истории и т.д.

---

321 «JIRA — Issue & Project Tracking Software» [https://www.atlassian.com/software/jira/]

322 «What is an Issue» [https://confluence.atlassian.com/display/JIRA/What+is+an+Issue]

323 «Defining Issue Type Field Values» [https://confluence.atlassian.com/display/JIRA/Defining+Issue+Type+Field+Values]

Рисунок 2.5.f — Создание отчёта о дефекте в JIRA

14. External issue id (идентификатор внешнего артефакта) позволяет связать отчёт о дефекте или иной артефакт с внешним документом.
15. Epic link (ссылка на историю/область) содержит ссылку на историю/область (см. пункт 13), наиболее близко относящуюся к дефекту.
16. Has a story/s (истории) содержит ссылки и/или описание пользовательских историй, связанных с дефектом (как правило, здесь приводятся ссылки на внешние документы).
17. Tester (тестировщик) содержит имя автора описания дефекта.
18. Additional information (дополнительная информация) содержит полезную дополнительную информацию о дефекте.
19. Sprint (спринт) содержит номер спринта (2–4-недельной итерации разработки проекта в терминологии гибких методологий управления проектами), во время которого был обнаружен дефект.

Многие дополнительные поля и возможности становятся доступными при других операциях с дефектами (просмотром или редактированием созданного дефекта, просмотре отчётов и т.д.)

Bugzilla[324]

1. Product (продукт) позволяет указать, к какому продукту (проекту) относится дефект.
2. Reporter (автор отчёта) содержит e-mail автора описания дефекта.
3. Component (компонент) содержит указание компонента приложения, к которому относится описываемый дефект.
4. Component description (описание компонента) содержит описание компонента приложения, к которому относится описываемый дефект. Эта информация загружается автоматически при выборе компонента.
5. Version (версия) содержит указание версии продукта, в которой был обнаружен дефект.
6. Severity (важность) содержит указание важности дефекта. По умолчанию предложены такие варианты:
    - Blocker (блокирующий дефект) — дефект не позволяет решить с помощью приложения некоторую задачу.
    - Critical (критическая важность).
    - Major (высокая важность).
    - Normal (обычная важность).
    - Minor (низкая важность).
    - Trivial (самая низкая важность).
    - Enhancement (предложение по улучшению) — было описано подробно в разделе, посвящённом полям отчёта о дефекте (см. описание поля «симптом», значение «предложение по улучшению»{176}).
7. Hardware (аппаратное обеспечение) позволяет выбрать профиль аппаратного окружения, в котором проявляется дефект.
8. OS (операционная система) позволяет указать операционную систему, под которой проявляется дефект.

[324] «Bugzilla» [https://www.bugzilla.org]

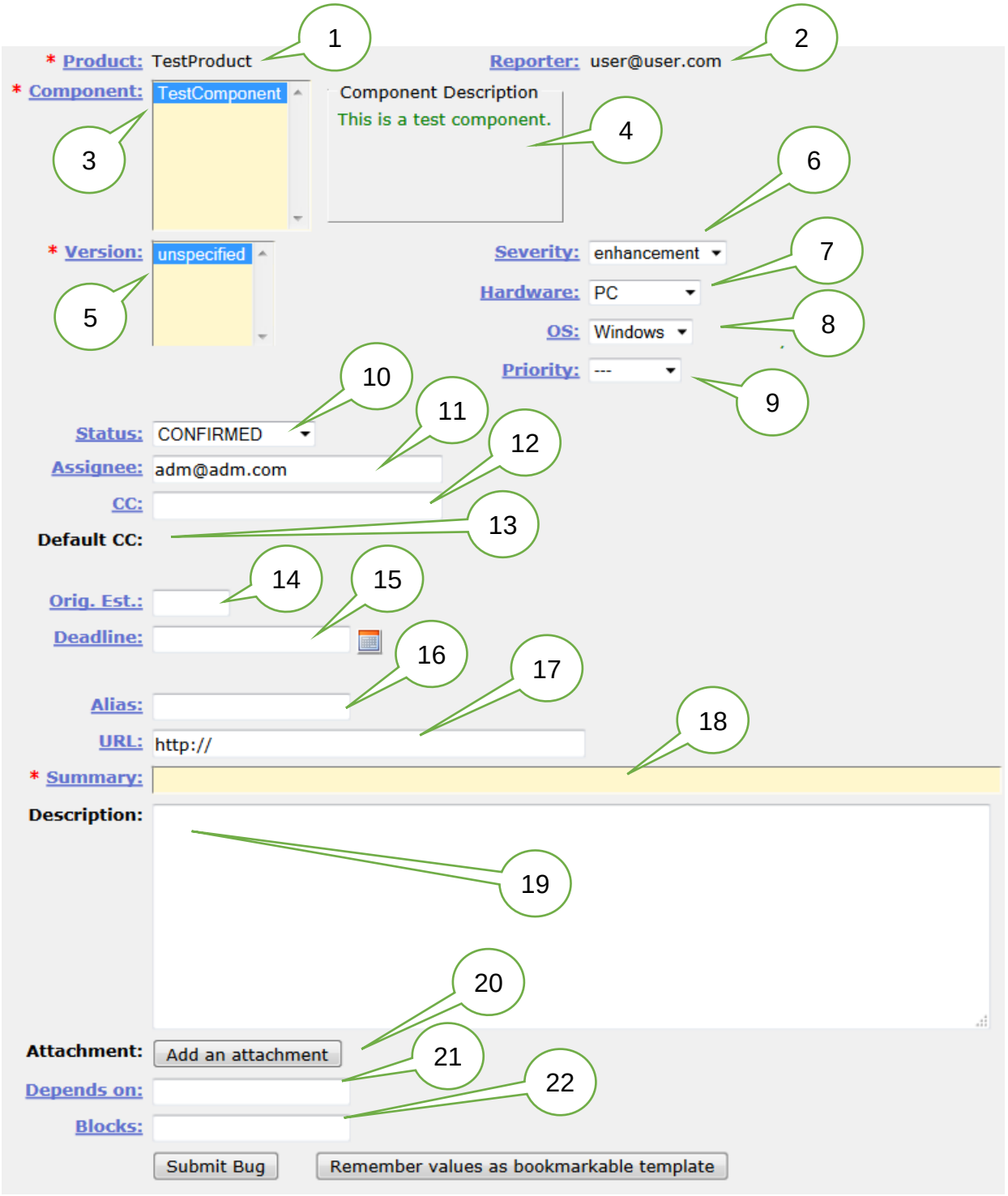

Рисунок 2.5.g — Создание отчёта о дефекте в Bugzilla

9. Priority (срочность) позволяет указать срочность исправления дефекта. По умолчанию Bugzilla предлагает следующие варианты:
    - Highest (самая высокая срочность).
    - High (высокая срочность).
    - Normal (обычная срочность).
    - Low (низкая срочность).
    - Lowest (самая низкая срочность).

10. Status (статус) позволяет установить статус отчёта о дефекте. По умолчанию Bugzilla предлагает следующие варианты статусов:
    - Unconfirmed (не подтверждено) — дефект пока не изучен, и нет гарантии того, что он действительно корректно описан.
    - Confirmed (подтверждено) — дефект изучен, корректность описания подтверждена.
    - In progress (в работе) — ведётся работа по изучению и устранению дефекта.

    В официальной документации рекомендуется сразу же после установки Bugzilla сконфигурировать набор статусов и правила жизненного цикла отчёта о дефектах в соответствии с принятыми в вашей компании правилами.
11. Assignee (ответственный) указывает e-mail участника проектной команды, ответственного за изучение и исправление дефекта.
12. CC (уведомлять) содержит список e-mail адресов участников проектной команды, которые будут получать уведомления о происходящем с данным дефектом.
13. Default CC (уведомлять по умолчанию) содержит e-mail адрес(а) участников проектной команды, которые по умолчанию будут получать уведомления о происходящем с любыми дефектами (чаще всего здесь указываются e-mail адреса рассылок).
14. Original estimation (начальная оценка) позволяет указать начальную оценку того, сколько времени займёт устранение дефекта.
15. Deadline (крайний срок) позволяет указать дату, к которой дефект обязательно нужно исправить.
16. Alias (псевдоним) позволяет указать короткое запоминающееся название дефекта (возможно, в виде некоей аббревиатуры) для удобства упоминания дефекта в разнообразных документах.
17. URL (URL) позволяет указать URL, по которому проявляется дефект (особенно актуально для веб-приложений).
18. Summary (краткое описание) позволяет указать краткое описание дефекта.
19. Description (подробное описание) позволяет указать подробное описание дефекта.
20. Attachment (вложение) позволяет добавить к отчёту о дефекте вложения в виде прикреплённых файлов.
21. Depends on (зависит от) позволяет указать перечень дефектов, которые должны быть устранены до начала работы с данным дефектом.
22. Blocks (блокирует) позволяет указать перечень дефектов, к работе с которыми можно будет приступить только после устранения данного дефекта.

Mantis[325]

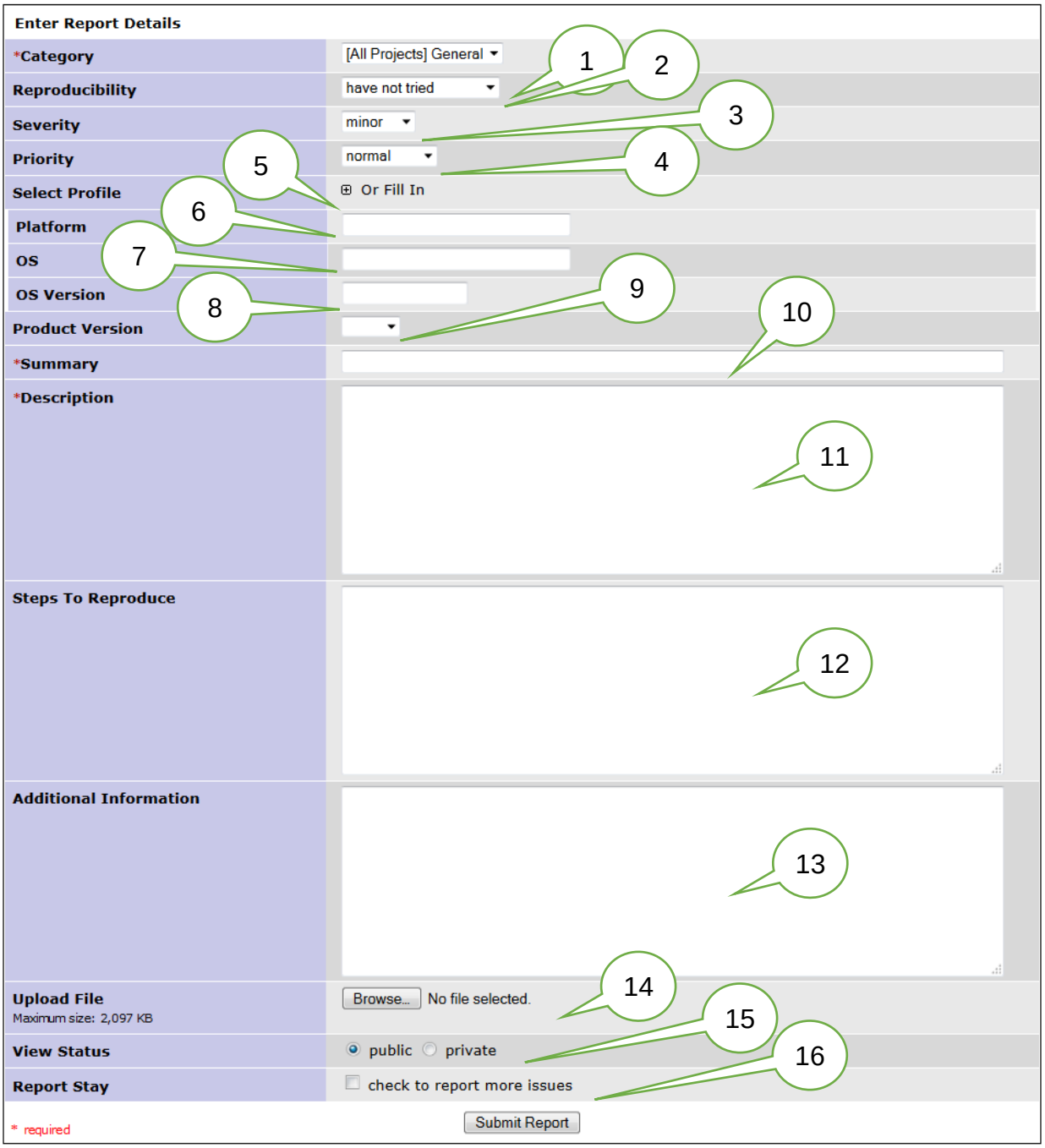

Рисунок 2.5.h — Создание отчёта о дефекте в Mantis

1. Category (категория) содержит указание проекта или компонента приложения, к которому относится описываемый дефект.
2. Reproducibility (воспроизводимость) дефекта. Mantis предлагает нетипично большое количество вариантов:
   - Always (всегда).
   - Sometimes (иногда).
   - Random (случайным образом) — вариация на тему «иногда», когда не удалось установить никакой закономерности проявления дефекта.
   - Have not tried (не проверено) — это не столько воспроизводимость, сколько статус, но Mantis относит это значение к данному полю.

[325] «Mantis Bug Tracker» [https://www.mantisbt.org]

- Unable to reproduce (не удалось воспроизвести) — это не столько воспроизводимость, сколько резолюция к отклонению отчёта о дефекте, но в Mantis тоже отнесено к данному полю.
- N/A (non-applicable, неприменимо) — используется для дефектов, к которым не применимо понятие воспроизводимости (например, проблемы с документацией).

3. Severity (важность) содержит указание важности дефекта. По умолчанию предложены такие варианты:
   - Block (блокирующий дефект) — дефект не позволяет решить с помощью приложения некоторую задачу.
   - Crash (критическая важность) — как правило, относится к дефектам, вызывающим неработоспособность приложения.
   - Major (высокая важность).
   - Minor (низкая важность).
   - Tweak (доработка) — как правило, косметический дефект.
   - Text (текст) — как правило, дефект относится к тексту (опечатки и т.д.).
   - Trivial (самая низкая важность).
   - Feature (особенность) — отчёт представляет собой не описание дефекта, а запрос на добавление/изменение функциональности или свойств приложения.
4. Priority (срочность) позволяет указать срочность исправления дефекта. По умолчанию Mantis предлагает следующие варианты:
   - Immediate (незамедлительно).
   - Urgent (самая высокая срочность).
   - High (высокая срочность).
   - Normal (обычная срочность).
   - Low (низкая срочность).
   - None (нет) — срочность не указана или не может быть определена.
5. Select profile (выбрать профиль) позволяет выбрать из заранее подготовленного списка профиль аппаратно-программной конфигурации, под которой проявляется дефект. Если такого списка нет или он не содержит необходимых вариантов, можно вручную заполнить поля 6–7–8 (см. ниже).
6. Platform (платформа) позволяет указать аппаратную платформу, под которой проявляется дефект.
7. OS (операционная система) позволяет указать операционную систему, под которой проявляется дефект.
8. OS Version (версия операционной системы) позволяет указать версию операционной системы, под которой проявляется дефект.
9. Product version (версия продукта) позволяет указать версию приложения, в которой был обнаружен дефект.
10. Summary (краткое описание) позволяет указать краткое описание дефекта.
11. Description (подробное описание) позволяет указать подробное описание дефекта.
12. Steps to reproduce (шаги по воспроизведению) позволяет указать шаги по воспроизведению дефекта.
13. Additional information (дополнительная информация) позволяет указать любую дополнительную информацию, которая может пригодиться при анализе и устранении дефекта.
14. Upload file (загрузить файл) позволяет загрузить копии экрана и тому подобные файлы, которые могут быть полезны при анализе и устранении дефекта.

15. View status (статус просмотра) позволяет управлять правами доступа к отчёту о дефекте и предлагает по умолчанию два варианта:
    - Public (публичный).
    - Private (закрытый).
16. Report stay (остаться в режиме добавления отчётов) — отметка этого поля позволяет после сохранения данного отчёта сразу же начать писать следующий.

Задание 2.5.b: изучите ещё 3–5 инструментальных средств управления жизненным циклом отчётов о дефектах, почитайте документацию по ним, посоздавайте в них несколько отчётов о дефектах.

2.5.5. Свойства качественных отчётов о дефектах

Отчёт о дефекте может оказаться некачественным (а следовательно, вероятность исправления дефекта понизится), если в нём нарушено одно из следующих свойств.

Тщательное заполнение всех полей точной и корректной информацией. Нарушение этого свойства происходит по множеству причин: недостаточный опыт тестировщика, невнимательность, лень и т.д. Самыми яркими проявлениями такой проблемы можно считать следующие:

- Часть важных для понимания проблемы полей не заполнена. В результате отчёт превращается в набор обрывочных сведений, использовать которые для исправления дефекта невозможно.
- Предоставленной информации недостаточно для понимания сути проблемы. Например, из такого плохого подробного описания вообще не ясно, о чём идёт речь: «Приложение иногда неверно конвертирует некоторые файлы».
- Предоставленная информация является некорректной (например, указаны неверные сообщения приложения, неверные технические термины и т.д.) Чаще всего такое происходит по невнимательности (последствия ошибочного copy-paste и отсутствия финальной вычитки отчёта перед публикацией).
- «Дефект» (именно так, в кавычках) найден в функциональности, которая ещё не была объявлена как готовая к тестированию. То есть тестировщик констатирует, что неверно работает то, что и не должно было (пока!) верно работать.
- В отчёте присутствует жаргонная лексика: как в прямом смысле — нелитературные высказывания, так и некие технические жаргонизмы, понятные крайне ограниченному кругу людей. Например: «Фигово подцепились чартники». (Имелось в виду: «Не все таблицы кодировок загружены успешно».)
- Отчёт вместо описания проблемы с приложением критикует работу кого-то из участников проектной команды. Например: «Ну каким дураком надо быть, чтобы такое сделать?!»
- В отчёте упущена некая незначительная на первый взгляд, но по факту критичная для воспроизведения дефекта проблема. Чаще всего это проявляется в виде пропуска какого-то шага по воспроизведению, отсутствию или недостаточной подробности описания окружения, чрезмерно обобщённом указании вводимых значений и т.п.
- Отчёту выставлены неверные (как правило, заниженные) важность или срочность. Чтобы избежать этой проблемы, стоит тщательно исследовать дефект, определять его наиболее опасные последствия и аргументированно отстаивать свою точку зрения, если коллеги считают иначе.
- К отчёту не приложены необходимые копии экрана (особенно важные для косметических дефектов) или иные файлы. Классика такой ошибки: отчёт описывает неверную работу приложения с некоторым файлом, но сам файл не приложен.
- Отчёт написан безграмотно с точки зрения человеческого языка. Иногда на это можно закрыть глаза, но иногда это становится реальной проблемой, например: «Not keyboard in parameters accepting values» (это реальная цитата; и сам автор так и не смог пояснить, что же имелось в виду).

Правильный технический язык. Это свойство в равной мере относится и к требованиям, и к тест-кейсам, и к отчётам о дефектах — к любой документации, а потому не будем повторяться — см. описанное ранее{131}.

Сравните два подробных описания дефекта:

Плохое подробное описание	Хорошее подробное описание
Когда мы как будто бы хотим убрать папку, где что-то внутри есть, оно не спрашивает, хотим ли мы.	Не производится запрос подтверждения при удалении непустого подкаталога в каталоге SOURCE_DIR. **Act:** производится удаление непустого подкаталога (со всем его содержимым) в каталоге SOURCE_DIR без запроса подтверждения. **Exp:** в случае, если в каталоге SOURCE_DIR приложение обнаруживает непустой подкаталог, оно прекращает работу с выводом сообщения «Non-empty subfolder [имя подкаталога] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.c.

Специфичность описания шагов. Говоря о тест-кейсах, мы подчёркивали, что в их шагах стоит придерживаться золотой середины между специфичностью и общностью. В отчётах о дефектах предпочтение, как правило, отдаётся специфичности по очень простой причине: нехватка какой-то мелкой детали может привести к невозможности воспроизведения дефекта. Потому, если у вас есть хоть малейшее сомнение в том, важна ли какая-то деталь, считайте, что она важна.

Сравните два набора шагов по воспроизведению дефекта:

Недостаточно специфичные шаги	Достаточно специфичные шаги
1. Отправить на конвертацию файл допустимого формата и размера, в котором русский текст представлен в разных кодировках. Дефект: конвертация кодировок производится неверно.	1. Отправить на конвертацию файл в формате HTML размером от 100 КБ до 1 МБ, в котором русский текст представлен в кодировках UTF8 (10 строк по 100 символов) и WIN-1251 (20 строк по 100 символов). Дефект: текст, который был представлен в UTF8, повреждён (представлен нечитаемым набором символов).

В первом случае воспроизвести дефект практически нереально, т.к. он заключается в особенностях работы внешних библиотек по определению кодировок текста в документе, в то время как во втором случае данных достаточно если и не для понимания сути происходящего (дефект на самом деле очень «хитрый»), то хотя бы для гарантированного воспроизведения и признания факта его наличия.

Ещё раз главная мысль: в отличие от тест-кейса отчёт о дефекте может обладать повышенной специфичностью, и это будет меньшей проблемой, чем невозможность воспроизведения дефекта из-за излишне обобщённого описания проблемы.

Отсутствие лишних действий и/или их длинных описаний. Чаще всего это свойство подразумевает, что не нужно в шагах по воспроизведению дефекта долго и по пунктам расписывать то, что можно заменить одной фразой:

Плохо	Хорошо
1. Указать в качестве первого параметра приложения путь к папке с исходными файлами. 2. Указать в качестве второго параметра приложения путь к папке с конечными файлами. 3. Указать в качестве третьего параметра приложения путь к файлу журнала. 4. Запустить приложение. Дефект: приложение использует первый параметр командной строки и как путь к папке с исходными файлами, и как путь к папке с конечными файлами.	1. Запустить приложение со всеми тремя корректными параметрами (особенно обратить внимание, чтобы SOURCE_DIR и DESTINATION_DIR не совпадали и не были вложены друг в друга в любом сочетании). Дефект: приложение прекращает работу с сообщением «SOURCE_DIR and DESTINATION_DIR may not be the same!» (судя по всему, первый параметр командной строки используется для инициализации имён обоих каталогов.)

Вторая по частоте ошибка — начало каждого отчёта о дефекте с запуска приложения и подробного описания по приведению его в то или иное состояние. Вполне допустимой практикой является написание в отчёте о дефекте приготовлений (по аналогии с тест-кейсами) или описание нужного состояния приложения в одном (первом) шаге.

Сравните:

Плохо	Хорошо
1. Запустить приложение со всеми верными параметрами. 2. Подождать более 30 минут. 3. Передать на конвертацию файл допустимого формата и размера. Дефект: приложение не обрабатывает файл.	Предусловие: приложение запущено и проработало более 30 минут. 1. Передать на конвертацию файл допустимого формата и размера. Дефект: приложение не обрабатывает файл.

Отсутствие дубликатов. Когда в проектной команде работает большое количество тестировщиков, может возникнуть ситуация, при которой один и тот же дефект будет описан несколько раз разными людьми. А иногда бывает так, что даже один и тот же тестировщик уже забыл, что когда-то давно уже обнаруживал некую проблему, и теперь описывает её заново. Избежать подобной ситуации позволяет следующий набор рекомендаций:

- Если вы не уверены, что дефект не был описан ранее, произведите поиск с помощью вашего инструментального средства управления жизненным циклом отчётов о дефектах.
- Пишите максимально информативные краткие описания (т.к. поиск в первую очередь проводят по ним). Если на вашем проекте накопится множество дефектов с краткими описаниями в стиле «Кнопка не работает», вы потратите очень много времени, раз за разом перебирая десятки таких отчётов в поисках нужной информации.
- Используйте по максимуму возможности вашего инструментального средства: указывайте в отчёте о дефекте компоненты приложения, ссылки на требования, расставляйте теги и т.д. — всё это позволит легко и быстро сузить в будущем круг поиска.
- Указывайте в подробном описании дефекта текст сообщений от приложения, если это возможно. По такому тексту можно найти даже тот отчёт, в котором остальная информация приведена в слишком общем виде.
- Старайтесь по возможности участвовать в т.н. «собраниях по прояснению»

(clarification meetings[326]), т.к. пусть вы и не запомните каждый дефект или каждую пользовательскую историю дословно, но в нужный момент может возникнуть ощущение «что-то такое я уже слышал», которое заставит вас произвести поиск и подскажет, что именно искать.

- Если вы обнаружили какую-то дополнительную информацию, внесите её в уже существующий отчёт о дефекте (или попросите сделать это его автора), но не создавайте отдельный отчёт.

Очевидность и понятность. Описывайте дефект так, чтобы у читающего ваш отчёт не возникло ни малейшего сомнения в том, что это действительно дефект. Лучше всего это свойство достигается за счёт тщательного объяснения фактического и ожидаемого результата, а также указания ссылки на требование в поле «Подробное описание».

Сравните:

Плохое подробное описание	Хорошее подробное описание
Приложение не сообщает об обнаруженных подкаталогах в каталоге SOURCE_DIR.	Приложение не уведомляет пользователя об обнаруженных в каталоге SOURCE_DIR подкаталогах, что приводит к необоснованным ожиданиям пользователями обработки файлов в таких подкаталогах. **Act:** приложение начинает (продолжает) работу, если в каталоге SOURCE_DIR находятся подкаталоги. **Exp:** в случае если в каталоге SOURCE_DIR приложение при запуске или в процессе работы обнаруживает пустой подкаталог, оно автоматически его удаляет (**логично ли это?**), если же обнаружен непустой подкаталог, приложение прекращает работу с выводом сообщения «Non-empty subfolder [имя подкаталога] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.c.

В первом случае после прочтения подробного описания очень хочется спросить: «И что? А оно разве должно уведомлять?» Второй же вариант подробного описания даёт чёткое пояснение, что такое поведение является ошибочным согласно текущему варианту требований. Более того, во втором варианте отмечен вопрос (а в идеале нужно сделать соответствующую отметку и в самом требовании), призывающий пересмотреть алгоритм корректного поведения приложения в подобной ситуации: т.е. мы не только качественно описываем текущую проблему, но и поднимаем вопрос о дальнейшем улучшении приложения.

Прослеживаемость. Из содержащейся в качественном отчёте о дефекте информации должно быть понятно, какую часть приложения, какие функции и какие требования затрагивает дефект. Лучше всего это свойство достигается правильным использованием возможностей инструментального средства управления отчётами о дефектах: указывайте в отчёте о дефекте компоненты приложения, ссылки на требования, тест-кейсы, смежные отчёты о дефектах (похожих дефектах; зависимых и зависящих от данного дефектах), расставляйте теги и т.д.

Некоторые инструментальные средства даже позволяют строить визуальные схемы на основе таких данных, что позволяет управление прослеживаемостью

[326] **Clarification meeting**. A discussion that helps the customers achieve “advance clarity” — consensus on the desired behavior of each story — by asking questions and getting examples. [«Agile Testing», Lisa Crispin, Janet Gregory]

даже на очень больших проектах превратить из непосильной для человека задачи в тривиальную работу.

Отдельные отчёты для каждого нового дефекта. Существует два незыблемых правила:

- В каждом отчёте описывается **ровно один** дефект (если один и тот же дефект проявляется в нескольких местах, эти проявления перечисляются в подробном описании).
- При обнаружении нового дефекта создаётся новый отчёт. **Нельзя** для описания **нового** дефекта править **старые** отчёты, переводя их в состояние «открыт заново».

Нарушение первого правила приводит к объективной путанице, которую проще всего проиллюстрировать так: представьте, что в одном отчёте описано два дефекта, один из которых был исправлен, а второй — нет. В какое состояние переводить отчёт? Неизвестно.

Нарушение второго правила вообще порождает хаос: мало того, что теряется информация о ранее возникавших дефектах, так к тому же возникают проблемы со всевозможными метриками и банальным здравым смыслом. Именно во избежание этой проблемы на многих проектах правом перевода отчёта из состояния «закрыт» в состояние «открыт заново» обладает ограниченный круг участников команды.

Соответствие принятым шаблонам оформления и традициям. Как и в случае с тест-кейсами, с шаблонами оформления отчётов о дефектах проблем не возникает: они определены имеющимся образцом или экранной формой инструментального средства управления жизненным циклом отчётов о дефектах. Но что касается традиций, которые могут различаться даже в разных командах в рамках одной компании, то единственный совет: почитайте уже готовые отчёты о дефектах, перед тем как писать свои. Это может сэкономить вам много времени и сил.

2.5.6. Логика создания эффективных отчётов о дефектах

При создании отчёта о дефекте рекомендуется следовать следующему алгоритму:

0. Обнаружить дефект ☺.
1. Понять суть проблемы.
2. Воспроизвести дефект.
3. Проверить наличие описания найденного вами дефекта в системе управления дефектами.
4. Сформулировать суть проблемы в виде «что сделали, что получили, что ожидали получить».
5. Заполнить поля отчёта, начиная с подробного описания.
6. После заполнения всех полей внимательно перечитать отчёт, исправив неточности и добавив подробности.
7. Ещё раз перечитать отчёт, т.к. в пункте 6 вы точно что-то упустили ☺.

Теперь — о каждом шаге подробнее.

Понять суть проблемы

Всё начинается именно с понимания происходящего с приложением. Только при наличии такого понимания вы сможете написать по-настоящему качественный отчёт о дефекте, верно определить важность дефекта и дать полезные рекомендации по его устранению. В идеале отчёт о дефекте описывает именно суть проблемы, а не её внешнее проявление.

Сравните два отчёта об одной и той же ситуации (приложение «Конвертер файлов» не различает файлы и символические ссылки на файлы, что приводит к серии аномалий в работе с файловой системой).

Плохой отчёт, при написании которого суть проблемы не понята:

Краткое описание	Подробное описание	Шаги по воспроизведению
Обрабатываются файлы вне SOURCE_DIR.	Иногда по непонятным причинам приложение обрабатывает случайные файлы вне каталога SOURCE_DIR. **Act:** обрабатываются отдельные файлы вне SOURCE_DIR. **Exp:** обрабатываются только файлы, находящиеся в SOURCE_DIR. Req: ДС-2.1.	К сожалению, не удалось обнаружить последовательность шагов, приводящих к появлению этого дефекта.

Воспроизводимость	Важность	Срочность	Симптом	Возможность обойти	Комментарий
Иногда	Высокая	Высокая	Некорректная операция	Нет	

Хороший отчёт, при написании которого суть проблемы понята:

Краткое описание	Подробное описание	Шаги по воспроизведению
Приложение не различает файлы и символические ссылки на файлы.	Если в каталог SOURCE_DIR поместить символическую ссылку на файл, возникает следующее ошибочное поведение: а) Если символическая ссылка указывает на файл внутри SOURCE_DIR, файл обрабатывается дважды, а в DESTINATION_DIR перемещается как сам файл, так и символическая ссылка на него. б) Если символическая ссылка указывает на файл вне SOURCE_DIR, приложение обрабатывает этот файл, перемещает символическую ссылку и сам файл в DESTINATION_DIR, а затем продолжает обрабатывать файлы в каталоге, в котором изначально находился обработанный файл. **Act:** приложение считает символические ссылки на файлы самими файлами (см. подробности выше). **Exp:** в случае если в каталоге SOURCE_DIR приложение обнаруживает символическую ссылку, оно прекращает работу с выводом сообщения «Symbolic link [имя символической ссылки] in SOURCE_DIR folder detected. Remove it manually or restart application with --force_file_operations key to remove automatically.» **Req:** UR.56.BF.4.e.	1. В произвольном месте создать следующую структуру каталогов: /SRC/ /DST/ /X/ 2. Поместить в каталоги SRC и X несколько произвольных файлов допустимого формата и размера. 3. Создать в каталоге SRC две символические ссылки: а) на любой из файлов внутри каталога SRC; б) на любой из файлов внутри каталога X. 4. Запустить приложение. Дефект: в каталог DST перемещены как файлы, так и символические ссылки; содержимое каталога X обработано и перемещено в каталог DST.

Воспроизводимость	Важность	Срочность	Симптом	Возможность обойти	Комментарий
Всегда	Высокая	Обычная	Некорректная операция	Нет	Быстрый взгляд на код показал, что вместо is_file() используется file_exists(). Похоже, проблема в этом. Также этот дефект приводит к попытке обработать каталоги как файлы (см. BR-999.99). В алгоритме обработки SOURCE_DIR явно есть логическая ошибка: ни при каких условиях приложение не должно обрабатывать файлы, находящиеся вне SOURCE_DIR, т.е. что-то не так с генерацией или проверкой полных имён файлов.

Воспроизвести дефект

Это действие не только поможет в дальнейшем правильно заполнить поле «Воспроизводимость[174]», но и позволит избежать неприятной ситуации, в которой за дефект приложения будет принят некий кратковременный сбой, который (скорее всего) произошёл где-то в вашем компьютере или в иной части ИТ-инфраструктуры, не имеющей отношения к тестируемому приложению.

Проверить наличие описания найденного вами дефекта

Обязательно стоит проверить, нет ли в системе управления дефектами описания именного того дефекта, который вы только что обнаружили. Это простое действие, не относящееся непосредственно к написанию отчёта о дефекте, но значительно сокращающее количество отчётов, отклонённых с резолюцией «дубликат».

Сформулировать суть проблемы

Формулировка проблемы в виде «что сделали (шаги по воспроизведению), что получили (фактический результат в подробном описании), что ожидали получить (ожидаемый результат в подробном описании)» позволяет не только подготовить данные для заполнения полей отчёта, но и ещё лучше понять суть проблемы.

В общем же случае формула «что сделали, что получили, что ожидали получить» хороша по следующим причинам:

- Прозрачность и понятность: следуя этой формуле, вы готовите именно данные для отчёта о дефекте, не скатываясь в пространные отвлечённые рассуждения.
- Лёгкость верификации дефекта: разработчик, используя эти данные, может быстро воспроизвести дефект, а тестировщик после исправления дефекта удостовериться, что тот действительно исправлен.
- Очевидность для разработчиков: ещё до попытки воспроизведения дефекта видно, на самом ли деле описанное является дефектом, или тестировщик где-то ошибся, записав в дефекты корректное поведение приложения.
- Избавление от лишней бессмысленной коммуникации: подробные ответы на «что сделали, что получили, что ожидали получить» позволяют решать проблему и устранять дефект без необходимости запроса, поиска и обсуждения дополнительных сведений.
- Простота: на финальных стадиях тестирования с привлечением конечных пользователей можно ощутимо повысить эффективность поступающей обратной связи, если объяснить пользователям суть этой формулы и попросить их придерживаться её при написании сообщений об обнаруженных проблемах.

Информация, полученная на данном этапе, становится фундаментом для всех дальнейших действий по написанию отчёта.

Заполнить поля отчёта

Поля отчёта о дефекте мы уже рассмотрели ранее[169], сейчас лишь подчеркнём, что начинать лучше всего с подробного описания, т.к. в процессе заполнения этого поля может выявиться множество дополнительных деталей, а также появятся мысли по поводу формулирования сжатого и информативного краткого описания.

Если вы понимаете, что для заполнения какого-то поля у вас не хватает данных, проведите дополнительное исследование. Если и оно не помогло, опишите в

соответствующем поле (если оно текстовое), почему вы затрудняетесь с его заполнением, или (если поле представляет собой список) выберите значение, которое, на ваш взгляд, лучше всего характеризует проблему (в некоторых случаях инструментальное средство позволяет выбрать значение наподобие «неизвестно», тогда выберите его).

Если у вас нет идей по поводу устранения дефекта, или он настолько тривиален, что не нуждается в подобных пояснениях, не пишите в комментариях «текст ради текста»: комментарии вида «рекомендую устранить этот дефект» не просто бессмысленны, но ещё и раздражают.

Перечитать отчёт (и ещё раз перечитать отчёт)

После того как всё написано, заполнено и подготовлено, ещё раз внимательно перечитайте написанное. Очень часто вы сможете обнаружить, что в процессе доработки текста где-то получились логические нестыковки или дублирование, где-то вам захочется улучшить формулировки, где-то что-то поменять.

Идеал недостижим, и не стоит тратить вечность на один отчёт о дефекте, но и отправлять невычитанный документ — тоже признак дурного тона.

После оформления отчёта о дефекте рекомендуется дополнительно исследовать ту область приложения, в которой вы только что обнаружили дефект. Практика показывает, что дефекты часто проявляются группами.

2.5.7. Типичные ошибки при написании отчётов о дефектах

Перед прочтением этого текста рекомендуется перечитать раздел с описанием типичных ошибок при составлении тест-кейсов и чек-листов{155}, т.к. многие описанные там проблемы актуальны и для отчётов о дефектах (ведь и то, и другое — специфическая техническая документация).

Ошибки оформления и формулировок

Плохие краткие описания (summary). Формально эта проблема относится к оформлению, но фактически она куда опаснее: ведь чтение отчёта о дефекте и осознание сути проблемы начинается именно с краткого описания. Ещё раз напомним его суть:

- Отвечает на вопросы «что?», «где?», «при каких условиях?».
- Должно быть предельно кратким.
- Должно быть достаточно информативным для понимания сути проблемы.

Посмотрите на эти краткие описания и попробуйте ответить себе на вопрос о том, что за проблема возникает, где она возникает, при каких условиях она возникает:

- «*Неожиданное прерывание*».
- «*Найдено 19 элементов*».
- «*Поиск по всем типам файлов*».
- «*Неинформативная ошибка*».
- «*В приложении красный шрифт*».
- «*Error when entering just the name of computer disk or folder*».
- «*No reaction to 'Enter' key*».

Иногда ситуацию спасает поле «подробное описание», из которого можно понять суть проблемы, но даже в этом случае сначала нужно приложить немало усилий, чтобы соотнести такое краткое описание с подробным, где представлено совершенно иное и иначе.

Перечитайте ещё раз раздел про формулировку качественных кратких описаний{170}.

Идентичные краткое и подробное описания (summary и description). Да, изредка бывают настолько простые дефекты, что для них достаточно одного краткого описания (например, «Опечатка в имени пункта главного меню “File” (сейчас “Fille”)»), но если дефект связан с неким более-менее сложным поведением приложения, стоит продумать как минимум три способа описания проблемы:

- краткий для поля «краткое описание» (его лучше формулировать в самом конце размышлений);
- подробный для поля «подробное описание» (поясняющий и расширяющий информацию из «краткого описания»);
- ещё один краткий для последнего шага в шагах по воспроизведению дефекта.

И это не интеллектуальные игры от переизбытка свободного времени, это вполне рабочий инструмент для формирования понимания сути проблемы (поверьте, вы едва ли сможете объяснить тремя разными способами то, что не понимаете).

Отсутствие в подробном описании явного указания фактического результата, ожидаемого результата и ссылки на требование, если они важны, и их представляется возможным указать.

Да, для мелочей наподобие опечаток в надписях этого можно не делать (и всё равно: при наличии времени лучше написать).

Но что можно понять из отчёта о дефекте, в кратком и подробном описании которого сказано только «приложение показывает содержимое OpenXML-файлов»? А что, не должно показывать? В чём вообще проблема? Ну, показывает и показывает — разве это плохо? Ах, оказывается (!), приложение должно не показывать содержимое этих файлов, а открывать их с помощью соответствующей внешней программы. Об этом можно догадаться из опыта. Но догадка — плохой помощник в случае, когда надо переписывать приложение, — можно сделать только хуже. Это также можно (наверное) понять, вдумчиво читая требования. Но давайте будем реалистами: отчёт о дефекте будет отклонён с резолюцией «описанное поведение не является дефектом» («not a bug»).

Игнорирование кавычек, приводящее к искажению смысла. Как вы поймёте такое краткое описание, как «запись исчезает при наведении мыши»? Какая-то запись исчезает при наведении мыши? А вот и нет, оказывается, «поле “Запись” исчезает при наведении мыши». Даже если не дописать слово «поле», кавычки подскажут, что имеется в виду имя собственное, т.е. название некоего элемента.

Общие проблемы с формулировками фраз на русском и английском языках. Да, нелегко сразу научиться формулировать мысль одновременно очень кратко и информативно, но не менее сложно читать подобные творения (цитаты дословные):

- «*Поиск не работает нажатием кнопки Enter*».
- «*По умолчанию в поле где искать стоит +*».
- «*При поиске файлов в обширном каталоге приложение ненадолго "зависает"*».
- «*При закрытии ошибки программа закрывается*».
- «*The application doesn't work with the from keyboard by the user in the field "What to search"*».

Лишние пункты в шагах воспроизведения. Не стоит начинать «с сотворения мира», большинство проектной команды достаточно хорошо знает приложение, чтобы «опознать» его ключевые части, потому — сравните:

Плохо	Хорошо
1. Запустить приложение. 2. Открыть пункт меню «Файл». 3. Выбрать пункт меню «Новый». 4. Заполнить текстом не менее трёх страниц. 5. Открыть пункт меню «Файл». 6. Открыть подпункт меню «Печать». 7. Открыть вкладку «Параметры печати». 8. Выбрать в списке «Двусторонняя печать» значение «Нет». 9. Распечатать документ на принтере, поддерживающем дуплексную печать. Дефект: печать по-прежнему двусторонняя.	1. Создать или открыть файл с тремя и более непустыми страницами. 2. Выбрать «Файл» -> «Печать» -> «Параметры» -> «Двусторонняя печать» -> «Нет». 3. Распечатать документ на принтере, поддерживающем дуплексную печать. Дефект: печать по-прежнему двусторонняя.

Копии экрана в виде «копий всего экрана целиком». Чаще всего нужно сделать копию какого-то конкретного окна приложения, а не всего экрана — тогда поможет Alt+PrintScreen. Даже если важно захватить больше, чем одно окно, практически любой графический редактор позволяет отрезать ненужную часть картинки.

Копии экрана, на которых не отмечена проблема. Если обвести проблемную область красной линией, это в разы повысит скорость и простоту понимания сути проблемы в большинстве случаев.

Копии экрана и иные артефакты, размещённые на сторонних серверах. Эта ошибка заслуживает особого упоминания: категорически запрещено использовать для прикрепления к отчёту о дефекте копий экрана и других файлов ставшие распространёнными в последнее время сервисы обмена изображениями, файлами и т.д. Причин здесь две:

- в большинстве случаев на размещение изображения или иного файла на таких сервисах есть ограничения по времени хранения и/или количеству обращений (скачиваний) — иными словами, через некоторое время файл может стать недоступным;
- размещение информации о проекте на сторонних сервисах является раскрытием конфиденциальной информации, права на которую принадлежат заказчику.

Потому для хранения любых подобных артефактов следует использовать саму систему управления дефектами. Если в силу неких причин таковая не используется, все вложения стоит разместить прямо в том документе, в котором вы описываете дефект (изображения можно разместить просто «в виде картинок», иные артефакты — в виде внедрённого документа).

Откладывание написания отчёта «на потом». Стремление сначала найти побольше дефектов, а уже потом их описывать приводит к тому, что какие-то важные детали (а иногда и сами дефекты!) забываются. Если «на потом» измеряется не минутами, а часами или даже днями, проектная команда не получает важную информацию вовремя. Вывод простой: описывайте дефект сразу же, как только обнаружили его.

Пунктуационные, орфографические, синтаксические и им подобные ошибки. Без комментариев.

Логические ошибки

Выдуманные дефекты. Одной из наиболее обидных для тестировщика причин отклонения отчёта о дефекте является так называемое «описанное поведение не является дефектом» («not a bug»), когда по какой-то причине корректное поведение приложения описывается как ошибочное.

Но ещё хуже, когда «якобы ожидаемое поведение» просто… придумывается из головы. То есть нигде в требованиях не сказано, что приложение должно что-то такое делать, но при этом создаётся отчёт о дефекте из-за того, что приложение этого не делает. И ладно бы это были некие спорные случаи или случайно попавшие в отчёты о дефектах предложения по улучшению, но нет — от приложения начинают требовать каких-то совершенно нелогичных, нерациональных, безумных вещей. Откуда это? Зачем? Просто не делайте так.

Отнесение расширенных возможностей приложения к дефектам. Самым ярким примером этого случая является описание как дефекта того факта, что приложение запускается под операционными системами, не указанными явно в списке поддерживаемых. Лишь в очень редких случаях при разработке неких системных утилит и тому подобного ПО, крайне зависящего от версии ОС и потенциально способного «поломать» неподдерживаемую, это можно считать ошибкой (с точки зрения общего здравого смысла такое приложение действительно должно показывать предупреждение или даже сообщение об ошибке и завершать работу на неподдерживаемой ОС). Но что плохого в том, что детская игрушка запустилась на ОС предыдущего поколения? Это реально проблема?! Сомнительно.

Неверно указанные симптомы. Это не смертельно, всегда можно подправить, но если изначально отчёты будут сгруппированы по симптомам, это создаёт множество раздражающих неудобств.

Чрезмерно заниженные (или завышенные) важность и срочность. С этой бедой достаточно эффективно борются проведением общих собраний и пересмотром отчётов о дефектах силами всей команды (или хотя бы нескольких человек), но если эти показатели занижены именно **чрезмерно**, есть высокая вероятность, что пройдёт очень много времени, прежде чем до такого отчёта просто дойдёт очередь на следующих собраниях по пересмотру.

Концентрация на мелочах в ущерб главному. Здесь стоит упомянуть хрестоматийный пример, когда тестировщик нашёл проблему, приводящую к краху приложения с потерей пользовательских данных, но записал её как косметический дефект (в сообщении об ошибке, которое «перед смертью» показывало приложение, была опечатка). Всегда думайте о том, как произошедшая с приложением неприятность повлияет на пользователей, какие сложности они могут из-за этого испытать, насколько это для них важно, — тогда шанс увидеть реальную проблему резко повышается.

Техническая безграмотность. Да, вот так безапелляционно и жёстко. В некоторых случаях она просто вызывает грустную улыбку, но в некоторых… Представьте себе такое краткое описание (оно же идентично продублировано в подробном, т.е. это и есть всё описание дефекта): «Количество найденных файлов не соответствует реальной глубине вложенности каталога». А что, должно? Это ведь почти то же самое, что «цвет кошки не соответствует её размеру».

Ещё несколько показательных примеров (это примеры из разных, никак не связанных между собой отчётов о дефектах):

- Краткое описание: «По умолчанию выбран каталог аудиозаписи». (На самом деле в выпадающем списке «Что искать» выбрано значение «Аудиофайлы».)
- Пояснение в подробном описании: «У каталога не может быть даты и времени создания». (Хм. Может.)
- Ожидаемый результат: «Приложение верно распознало неподдерживаемую файловую систему и показало список файлов». (Ого! С этим приложением, скорее всего, можно будет и на философские темы побеседовать, если оно способно на такую магию.)

Указание в шагах воспроизведения информации, неважной для воспроизведения ошибки. Стремление прописать всё максимально подробно иногда принимает нездоровую форму, когда в отчёт о дефекте начинает попадать чуть ли не информация о погоде за окном и курс национальной валюты. Сравните:

Плохо	Хорошо
1. Создать на диске «J:» каталог «Data». 2. Разместить в созданном каталоге «Data» прилагаемые файлы «song1.mp3» размером 999.99 Kb и «song2.mp3» размером 888.88 Kb. 3. В поле «Где искать» указать «J:\Data». 4. В выпадающем списке «Что искать» выбрать «Аудиофайлы». 5. Нажать кнопку «Искать». Дефект: указанные в пункте 2 файлы не найдены.	1. В произвольном месте на локальном диске разместить один (или более) файл с расширением «.mp3». 2. Установить параметры поиска («Что искать» -> «Аудиофайлы», «Где искать» -> место расположения файла(ов) из пункта 1). 3. Произвести поиск. Дефект: приложение не обнаруживает файлы с расширением «.mp3».

Действительно ли для воспроизведения дефекта важно, чтобы поиск производился на диске «J:»? Действительно ли важно производить поиск именно файлов с такими именами и размерами в таком каталоге? Возможно, в некоем бесконечно маловероятном случае и да, тогда вопрос снимается. Но, скорее всего, это совершенно не важно, и нет никакой необходимости записывать эту информацию. Для большей уверенности можно провести дополнительное исследование.

Отсутствие в шагах воспроизведения информации, важной для воспроизведения дефекта. Нет, мы не издеваемся, этот пункт действительно строго противоположен предыдущему. Дефекты бывают разными. Очень разными. Иногда какой-то ключевой «мелочи» не хватит, чтобы разработчику удалось воспроизвести дефект или даже просто понять его суть. Несколько реальных примеров (подчёркнуты те детали, отсутствие которых долго не позволяло воспроизвести дефект):

- Приложение не сохраняло пользовательские настройки в случае, если в пути к каталогу для их сохранения были пробелы (два и более подряд пробела).
- Приложение некорректно завершало работу при открытии файлов, размер которых не позволял прочитать их целиком в оперативную память, доступный объём которой, в свою очередь, определяется значением параметра memory_limit в настройках среды исполнения.
- Приложение отображало неверную статистику работы пользователей, если в системе был хотя бы один пользователь, роль которого не была явно указана (NULL-значение в таблице БД, неверная работа подзапроса).

Как понять, насколько подробно надо описывать такие детали? Исследованием. Мало просто обнаружить некий единичный случай неверного поведения приложения, нужно понять закономерность такого поведения и его источник. Тогда необходимая степень детализации становится очевидной.

Сюда же можно отнести пресловутую воспроизводимость «иногда». Нужно продолжать искать причины, смотреть код, консультироваться с коллегами, проводить дополнительные тесты, исследовать похожую функциональность в других частях приложения, исследовать аналогичные приложения, «гуглить» и т.д. и т.п. Да, часть дефектов оказывается сильнее даже самых упорных тестировщиков, но вот процент таких дефектов можно очень сильно приблизить к нулю.

Игнорирование т.н. «последовательных дефектов». Иногда один дефект является следствием другого (допустим, файл повреждается при передаче на сервер, а затем приложение некорректно обрабатывает этот повреждённый файл). Да, если файл будет передан без повреждений, второй дефект может не проявиться. Но может и проявиться в другой ситуации, т.к. проблема никуда не исчезла: приложение некорректно обрабатывает повреждённые файлы. Потому стоит описать оба дефекта.