# ***Блок 1***

# Определение бага

**Баг** (bug) – это отклонение фактического результата (actual result) от ожидаемого результата (expected result).

## Условия жизни и процветания бага

1. Известен фактический результат.
2. Известен ожидаемый результат.
3. Известно, что фактический результат не соответствует ожидаемому.

## Понятие тестирования

**Тестирование** – это поиск багов. Поиск багов осуществляется следующим образом:

1. Узнаем (или уже знаем) ожидаемый результат.
2. Узнаем (или уже знаем) фактический результат.
3. Сравниваем пункт 1 с пунктом 2.

## Источники ожидаемого результата

**Спецификация** (spec) – это детальное описание того, как должно работать ПО. Таким образом, баг – это отклонение от спецификации.

## Функциональные баги и баги спека

**Функциональный баг** (functional bug) – это баг, когда фактическая работа кода не соответствует функциональному спеку.

Если в спеке не указано о тексте ошибки – это **баг в спецификации** (spec bug). В таком случае Вам необходимо отправить письмо **продюсеру** (создателю спеков) с целью корректировки спека. Если же программист написал нечто противоречащее здравому смыслу, необходимо рапортовать баг. Также следует проявлять инициативу в усовершенствовании спецификации (например, при указании возможности загрузки одной фотографии за раз порекомендовать о реализации возможности загрузки нескольких).

# ***Блок 2***

# Цель тестирования (Decoded)

**Цель тестирования** – это нахождение багов до того времени, как они будут обнаружены пользователями.

Концепции, наносящие вред тестировщику:

1. Цель тестирования – 100-процентная проверка ПО.
2. Критерий эффективности тестирования – количество багов, найденных до релиза.

Передача ПО пользователю – **релиз** (release).

Один из критериев качества кода – количество багов на 1000 строк кода. Количество багов, найденных до релиза, ничего не говорит об эффективности тестирования.

После каждого релиза данные о багах, найденных после релиза, классифицируются по заданным критериям и анализируются на предмет нахождения слабого звена в процессе разработки ПО.

Quality Assurance (**обеспечение качества**) – это забота о качестве в виде предотвращения (превентирования) появления багов.

Тестирование – это забота о качестве в виде обнаружения багов до того, как их найдут пользователи.

**Общее между QA и тестированием** – направлены на улучшение ПО.

Отличие QA и тестирования:

1. QA направлено на улучшение ПО посредством улучшения процесса его разработки.
2. Тестирование направлено на улучшение ПО посредством обнаружения багов.

Инженер по качеству – QA Engineer, тестировщик – Test Engineer.

# ***Блок 3***

# Понятие тест-кейса

**Тест-комплект** (test suite) – объединение тест-кейсов.

**Создание тест-кейса** (test case generation)

**Исполнение тест-кейса** (test case execution)

**Тест-кейс** (test case) – в переводе – тестируемая ситуация или оболочка для теста. Главной и неотъемлемой частью тест-кейса является ожидаемый результат. Тест-кейс может полностью состоять только из ожидаемого результата.

Для нахождения бага требуется как ожидаемый, так и фактический результат. В результате возникает потребность, как прийти к фактическому результату.

Исходя из основной компьютерной концепции «ввод-вывод» («input-out»):

* **шаги** – инструкция по вводу,
* **исполнение шагов** – ввод,
* **ожидаемый результат** – ожидаемый вывод,
* **фактический результат** – фактический вывод.

Исполнение тест-кейса завершается сравнением вывода фактического и вывода ожидаемого.

**Тест-кейс** – это инструмент тестировщика, предназначенный для записи и проверки одного и более ожидаемых результатов.

**Шаги** (procedure) – это часть тест-кейса, ведущая исполнителя тест-кейса к фактическому результату (выводу).

## Исход исполнения тест-кейса (test case result)

Каждый исполненный тест-кейс дает нам одно из двух:

1. **Положительный исход** (PASS), если фактический результат равен ожидаемому результату.
2. **Отрицательный исход** (FAIL), если фактический результат не равен ожидаемому, т.е. найден баг.

Иногда возникает ситуация, когда мы не можем пройти все шаги тест-кейса, т.к. мы заблокированы (test case execution is blocked). В данном случае мы рапортируем баг об отсутствии чего-то (например, кнопки «Завершить заказ») и откладываем исполнение тест-кейса до устранения бага.

## Полезные атрибуты тест кейса

**Уникальный ID** (Unique ID). ID должен быть уникальным в пределах всего департамента качества. *Приоритет тест-кейса* – это важность тест-кейса. Важность отражается по шкале от 1 до n, где 1 – высший результат, а n – низший. Рациональным является создание 4-х приоритетов.

**Идея** (Idea) – это описание конкретной вещи, проверяемой тест-кейсом (например, «Оплата может быть произведена картой VISA»).

**Подготовительная часть** (SETUP and ADDITIONAL INFO). Она может включать данные о существующем аккуанте пользователя или инструкцию по созданию нового аккуанта, атрибуты используемой кредитной карты, комментарии в помощь тестировщику и т.д.

**История редактирования** (Revision History)

Атрибуты истории редактирования:

* **Created on** <date> by <name> – Тест-кейс создан <дата> <кем>;
* **Modified on** <date> by <name> – Тест-кейс изменен <дата> <кем>;
* **Change** – Что, зачем и почему было изменено. В наших примерах мы не указываем само слово «change», а просто заполняем значение этого атрибута в поле справа от «Created on» или «Modified on».

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0001 | | 1 |
| **IDEA:** Оплата может быть произведена картой VISA  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Наименование товара: book117  Данные карты:  Номер: 9999-5148-2222-1277  Окончание действия: 12/07  CVV2: 778  SQL1: select result from cc\_transaction where id=<номер заказа>; | | | |
| **Revision History** | | | |
| **Created on:** 07/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 2. Введи имя пользователя. 3. Введи пароль. 4. Нажми кнопку «Войти». 5. Введи наименование товара в поле поиска. 6. Нажми кнопку «Найти». 7. Кликни линк «Добавить в корзину». 8. Кликни линк «Корзина». 9. Кликни линк «Оплатить». 10. Выбери вид карты. 11. Введи номер карты. 12. Введи срок окончания действия. 13. Введи CVV2. 14. Нажми кнопку «Завершить заказ». 15. Запиши номер заказа. | | > **«10»** | |
| 1. Запроси базу данных с SQL1 и запиши результат. | |

Такой вид тест-кейса называется **data-driven** («управляемый данными»), т.е. когда данные и инструкции по их применению не смешаны, а разделены и слинкованы. Полезность такого тест-кейса заключается в том, что на основе его сценария мы можем протестировать и другие аналогичные тест-кейсы. Для этого требуется лишь модификация данных.

**Поддерживаемость** (maintainablility) – насколько легко и просто можно изменить тест-кейс в случае изменений в ПО.

Разбиваем шаги нашего нового тест-кейса с картой на логические модули:

1. Вход в систему (логин – log in).
2. Поиск товара.
3. Добавление товара в корзину.
4. Оплата.
5. Фиксация номера заказа.
6. Запрос базы данных.

QA Knowledge Base – **Пособие для тестировщиков.**

Обобщим уже известные нам мероприятия по улучшению поддерживаемости тест-кейса:

1. Сделать тест-кейс data-driven.
2. Не описывать шаги по явно очевидным сценариям (например, логин).
3. Не давать конкретных деталей, если они не играют роли при исполнении тест-кейса (например, имя товара).
4. Вынести во внешний документ повторяющиеся сценарии (например, 7 шагов оплаты).

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0001 | | 1 |
| **IDEA:** Оплата может быть произведена картой VISA  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Наименование товара: book117  Данные карты:  Номер: 9999-5148-2222-1277  Окончание действия: 12/07  CVV2: 778  SQL1: select result from cc\_transaction where id=<номер заказа>; | | | |
| **Revision History** | | | |
| **Created on:** 07/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Modified on:** 09/03/2015 by В. Изотов | | Шаги были упрощены, чтобы сделать тест-кейс более удобным для поддержки | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 2. Войди в систему. 3. Найди любой товар. 4. Добавь товар в корзину. 5. Произведи оплату картой из секции SETUP and ADDITIONAL INFO. 6. Запиши номер заказа. | | > **«10»** | |
| 1. Запроси базу данных с SQL1 и запиши результат. | |

**Сколько ожидаемых результатов может быть в одном тест-кейсе?**

В идеальном варианте один, однако бывают случаи, когда требуется выполнение двух условий:

1. Значение «10» в соответствующей колонке соответствующей строки в базе данных.
2. Уменьшение баланса на счете с картой VISA на сумму, равную сумме оплаты.

В итоге мы имеем два пути:

1. Разложить идею тест-кейса на две идеи и создать два тест-кейса.
2. Оставить идею тест-кейса неприкосновенной и включить в один тест-кейс два ожидаемых результата. Это означает, что тест-кейс будет иметь положительный исход при совпадении обоих фактических результатов с ожидаемыми.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0001 | | 1 |
| **IDEA:** Оплата может быть произведена картой VISA  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Данные карты:  Номер: 9999-5148-2222-1277  Окончание действия: 12/07  CVV2: 778  SQL1: select result from cc\_transaction where id=<номер заказа>;  Баланс счета карты можно посмотреть здесь: www.main.testshop.rs/1277/balance.htm | | | |
| **Revision History** | | | |
| **Created on:** 07/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Modified on:** 09/03/2015 by В. Изотов | | Шаги были упрощены, чтобы сделать тест-кейс более удобным для поддержки | |
| **Modified on:** 09/03/2015 by В. Изотов | | Изменение шагов и второй ожидаемый результат с целью удостоверения в снятии денег со счета | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Запиши баланс счета карты. 2. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 3. Войди в систему. 4. Найди любой товар. 5. Добавь товар в корзину. 6. Произведи оплату картой из секции SETUP and ADDITIONAL INFO (!!! Запиши полную сумму заказа:). 7. Запиши номер заказа. 8. Запроси базу данных с SQL1 и запиши результат. | | S> **«10»** | |
| 1. Запиши баланс счета карты | | > Шаг 1 – Шаг 6 | |

В данном случае: прошли 8 шагов, остановились, проверили. Затем прошли девятый шаг, остановились, проверили.

Исход исполнения этого тест-кейса будет считаться положительным только при одновременной истинности двух условий:

1. Фактический результат после исполнения шага 8 = «10» и;
2. Фактический результат после исполнения шага 9 = Шаг 1 – Шаг 6 (то есть значение Шага 1 минус значение Шага 6).

В теории лучше было бы разбить нашу идею тест-кейса на две части и создать два отдельных тест-кейса:

1. IDEA: «Правильное значение вставляется в базу данных при использовании VISA»;
2. IDEA: «Верная сумма списывается с баланса карты».

Если есть возможность, лучше сделать два тест-кейса, однако с целью экономии времени или при его отсутствии имеет смысл включить в тест-кейс два или более ожидаемых решений.

Существует много случаев, когда несколько ожидаемых результатов просятся в один тест-кейс. В итоге нам нужно проверить:

* значение (-я) на веб-странице и;
* значение (-я) в базе данных.

Из этого следует, что нам нужна проверка **снаружи** и **изнутри** или на **front and** и **back end.**

Front end – интерфейс пользователя: текст, картинки, кнопки, линки и т.д.

Back end – ПО и данные, находящиеся за фасадом фронт-энда: HTML-код веб-страницы, веб-сервер, код приложения, база данных и т.д.

В нашем примере мы «разговаривали»:

* с фронт-энд ом – в шаге 5, когда добавляли товар в корзину;
* с бэк-эндом – в шаге 8, когда запрашивали базу данных.

## Проблемные тест-кейсы

Недостатки, которые следует удалять из тест-кейсов:

1. Зависимость тест-кейсов друг от друга.
2. Нечеткая формулировка шагов.
3. Нечеткая формулировка идеи и (или) ожидаемого результата.

Характеристика хорошего тест-кейса:

1. Отсутствие ссылок на другие тест-кейсы.
2. Независимость от «следов», оставленных другими тест-кейсами в вашем ПО или базе данных.

Перечисляющиеся в тест-кейсе шаги должны быть объективно четкими и ясными.

Нюансы для идеи тест-кейса и ожидаемого результата:

1. Не рекомендуется отсылка к внешнему документу.
2. Суть секции IDEA – это ОБЪЯСНЕНИЕ идеи тест-кейса.
3. Ожидаемый результат – это информация, на основании которой (вместе с фактическим результатом) мы принимаем решение об исходе тест-кейса.

## Тест-комплекты

**Тест-комплект** (test case suite) – это совокупность тест-кейсов, которые проверяют определенную часть нашего интернет-проекта и (или) определенный спек.

Создание нового файла с новым тест-комплектом обусловлено тем, что новые тест-кейсы всегда исполняются в первую очередь и нам просто удобно хранить их отдельно от старых.

Последовательность действий тестировщика:

1. Получаем новый спек.
2. Создаем новый файл, в котором создаем новые тест-кейсы для этого нового спека.
3. Исполняем новые тест-кейсы с их одновременной модификацией.
4. Если имеет смысл, переносим тест-кейсы в основной тест-комплект, где хранятся тест-кейсы, проверяющие ту же функциональную часть вашего интернет-проекта.

В первый раз тест-кейсы должны исполняться их автором, задача которого:

1. В случае необходимости добавить новые тест-кейсы.
2. В случае необходимости внести изменения по существу, например, если при создании тест-кейса тестировщик неправильно понял спек.
3. Если возможно, удалить лишние тест-кейсы, например, если два тест-кейса проверяют одну и ту же идею, дублируя друг друга.
4. Сделать тест-кейсы более удобными для поддержки.
5. Отшлифовать тест-кейсы, то есть сделать формулировки кристально-сверкающе-искристо ясными и точными.

«Шапка», которую можно разместить поверх тест-кейсов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Author:** | Spec ID: | Priority: | Producer: | **Developer:** |
| OVERVIEW: | | | | |
| GLOBAL SETUP and ADDITIONAL INFO: | | | | |

Author – автор тест-кейсов.

Spec ID – номер (или иной уникальный ID) спека. Сам ID должен быть линком к спеку в локальной сети.

Priority – приоритет тест-комплекта (например, от 1 до 4), обычно соответствующий приоритету спека.

Producer – продюсер, написавший спек.

Developer – программист, пишущий код в соответствии со спеком.

Секция OVERVIEW – краткий рассказ о том, чему посвящен данный тест-комплект.

Секция GLOBAL SETUP and ADDITIONAL INFO аналогична секции тест-кейса SETUP and ADDITIONAL INFO, только здесь мы говорим о повторяющихся вещах, которые будем использовать в более чем одном тест-кейсе, и вообще о любой другой полезной информации для всего тест-комплекта.

Вот так будет выглядеть файл credit\_card\_payments.doc, включающий тест-комплект «Покупка с использованием кредитных карт».

**Покупка с использованием кредитных карт (TS7122)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Author:**  В. Изотов | Spec ID:  1211 | Priority:  1 | Producer:  П. Хрипунов | **Developer:**  Н. Назаров |
| OVERVIEW:  Данный тест-комплект проверяет оплату картами VISA и MasterCard | | | | |
| GLOBAL SETUP and ADDITIONAL INFO:   1. SQL1: select result from cc\_transaction where id=<номер заказа>; 2. Баланс счета карты можно посмотреть здесь:   [www.main.testshop.rs/<четыре\_последних\_цифры\_карты>/balance.htm](http://www.main.testshop.rs/%3cчетыре_последних_цифры_карты%3e/balance.htm) | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | **CCPG0001** | | 1 |
| **IDEA:** Оплата может быть произведена картой VISA  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Данные карты:  Номер: 9999-5148-2222-1277  Окончание действия: 12/07  CVV2: 778 | | | |
| **Revision History** | | | |
| **Created on:** 07/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Modified on:** 09/03/2015 by В. Изотов | | Шаги были упрощены, чтобы сделать тест-кейс более удобным для поддержки | |
| **Modified on:** 09/03/2015 by В. Изотов | | Изменение шагов и второй ожидаемый результат с целью удостоверения в снятии денег со счета | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Запиши баланс счета карты. 2. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 3. Войди в систему. 4. Найди любой товар. 5. Добавь товар в корзину. 6. Произведи оплату картой из секции SETUP and ADDITIONAL INFO (!!! Запиши полную сумму заказа:). 7. Запиши номер заказа. 8. Запроси базу данных с SQL1 и запиши результат. | | > **«10»** | |
| 1. Запиши баланс счета карты | | > Шаг 1 – Шаг 6 | |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | **CCPG0002** | | 1 |
| **IDEA:** Оплата может быть произведена картой MasterCard  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Данные карты:  Номер: 3333-7112-4444-7844  Окончание действия: 12/10  CVV2: 676 | | | |
| **Revision History** | | | |
| **Created on:** 07/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Modified on:** 09/03/2015 by В. Изотов | | Шаги были упрощены, чтобы сделать тест-кейс более удобным для поддержки | |
| **Modified on:** 09/03/2015 by В. Изотов | | Изменение шагов и второй ожидаемый результат с целью удостоверения в снятии денег со счета | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Запиши баланс счета карты. 2. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 3. Войди в систему. 4. Найди любой товар. 5. Добавь товар в корзину. 6. Произведи оплату картой из секции SETUP and ADDITIONAL INFO (!!! Запиши полную сумму заказа:). 7. Запиши номер заказа. 8. Запроси базу данных с SQL1 и запиши результат. | | > **«20»** | |
| 1. Запиши баланс счета карты | | > Шаг 1 – Шаг 6 | |

Наш менеджер дает нам для проработки и создания тест-кейсов новый спек продюсера М. Чучикова: #1422 «Покупка с использованием Switch». Мы создаем новый файл: switch\_payments.doc. После того как мы его исполнили и причесали наши новые тест-кейсы (в данном случае один тест-кейс), получаем:

**Покупка с использованием Switch (TS7131)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Author:**  В. Изотов | Spec ID:  1422 | Priority:  1 | Producer:  М. Чучиков | **Developer:**  Н. Назаров |
| OVERVIEW:  Данный тест-комплект проверяет оплату картой Switch | | | | |
| GLOBAL SETUP and ADDITIONAL INFO:   1. SQL1: select result from cc\_transaction where id=<номер заказа>; 2. Баланс счета карты можно посмотреть здесь:   [www.main.testshop.rs/<четыре\_последних\_цифры\_карты>/balance.htm](http://www.main.testshop.rs/%3cчетыре_последних_цифры_карты%3e/balance.htm) | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | **SWPL0001** | | 1 |
| **IDEA:** Оплата может быть произведена картой Switch  SETUP and ADDITIONAL INFO:  Аккуант: testuser1/password  Данные карты:  Номер: 4444-1988-4444-5699  Окончание действия: 12/05  CVV2: 451 | | | |
| **Revision History** | | | |
| **Created on:** 09/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Запиши баланс счета карты. 2. Открой [www.main.testshop.rs](http://www.main.testshop.rs) 3. Войди в систему. 4. Найди любой товар. 5. Добавь товар в корзину. 6. Произведи оплату картой из секции SETUP and ADDITIONAL INFO (!!! Запиши полную сумму заказа:). 7. Запиши номер заказа. 8. Запроси базу данных с SQL1 и запиши результат. | | > **«30»** | |
| 1. Запиши баланс счета карты | | S\* > Шаг 1 – Шаг 6 | |

Теперь нам нужно просто объединить оба файла. Таким образом, у нас получился all new credit\_card\_payments.doc.

Покупка с использованием кредитных карт

|  |  |  |
| --- | --- | --- |
| **Часть 1.** Тестирование с Visa и MasterCard  **Часть 2.** Тестирование со Switch | | |
| **Часть 1** | | |
| **TC ID/Priority** | **CCPG0001** | 1 |
| **TC ID/Priority** | **CCPG0002** | 1 |
| **Часть 2** | | |
| **TC ID/Priority** | **SWPL0001** | 1 |

Генерация уникального ID тест-кейса может происходить:

* автоматически (для этого может быть написана простая программа),
* вручную (необходимо заключить конвенцию внутри департамента качества).

Процесс присвоения ID (на примере):

1. Пишем тест-кейсы. ID не прописываем.
2. «Обкатываем» их при первом исполнении с удалением тех из них, которые недостойны быть частью нашего тест-комплекта, и добавлением тех, которые пришли на ум по мере исполнения.
3. Присваиваем оставшимся тест-кейсам ID.

## Состояния тест-кейса

**Новый** (New). Первая редакция тест-кейса: «Created on:».

**Измененный** (Modified). Модификации, как правило, связаны с изменением спека или с улучшением тест-кейса. «Modified on:».

**Более недействителен** (Retired). Данная ситуация наступает в случае прекращения работы тестируемой вещи (функциональности, элемента пользовательского интерфейса) или в других случаях, например, когда тест-кейс дублирует другой.

В таком случае нам необходимо:

1. Создать специальную директорию в месте хранения файлов с тест-комплектами и назвать ее retired testcases.
2. Создаем в этой директории файл с тем же именем, что и файл тест-комплекта, из которого удаляем тест-кейс.
3. Вырезаем тест-кейс из файла, больше не нуждающегося в этих услугах, в одноименный файл директории retired testcases.

*Практическое задание*

Написать тест-кейс с идеей «Я могу убедить свою жену в чем угодно» и ожидаемым результатом «Дорогой, поезжайте с Алексеем на рыбалку. Вы так редко с ним видитесь».

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | **RWHS0001** | | 1 |
| **IDEA:** Я могу убедить свою жену в чем угодно  SETUP and ADDITIONAL INFO:  Объект коммуникации: Жена  Предлог воздействия:   1. Комплимент. 2. Усталость. 3. Отдых. 4. Рыбалка.   Член компании: сын Алексей.  Период отсутствия коммуникации с сыном: 2 недели | | | |
| **Revision History** | | | |
| **Created on:** 12/03/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Подойти к жене. 2. Сказать ей комплимент. 3. Сказать, что испытываешь усталость. 4. Сказать, что испытываешь желание отправиться на отдых. 5. Сказать, что хочешь поехать на рыбалку. 6. Сказать, что хочешь взять с собой сына Алексея. 7. Сказать, что не виделся с ним 2 недели. | | > **«Дорогой, поезжайте с Алексеем на рыбалку. Вы так редко с ним видитесь»** | |
| 1. Получить ответ | | > Шаг 1 – Шаг 7 | |

# ***Блок 4***

# Цикл разработки ПО

**Цикл (процесс) разработки ПО** – это путь от идеи до поддержки готового продукта.

Цикл разработки ПО «Waterfall»:

1. Идея.
2. Разработка дизайна продукта и создание документации.
3. Кодирование (создание кода).
4. Исполнение тестирования и ремонт багов.
5. Релиз.

### Идея

Идеи компонуются в MRD (Marketing Requirements Document) – документ о требованиях маркетинга, суть которого «хотелось бы иметь».

### Разработка дизайна проекта и создание спека

Разница между дизайном и идеей заключается в том, что идея – это описание цели, а дизайн – это описание пути к достижению цели.

Дизайн могут разрабатывать PMs (Product Managers), Producers или Product Designer. В результате появляются спеки, которые также называются PRD (Product Requirements Document) – документ о требованиях для продукта. Также его называют requirements (требования).

Спеки должны иметь уникальное название и уникальный ID, а также приоритет (1 (П1) – самый приоритетный спек).

Признаки хорошего спека:

1. Акцент на деталях и их четкое определение (например, не указано, что регистрации с двумя @@ в адресе электронной почты быть не может).
2. Забота о недопущении неверного толкования (например, тестировщик и программист должны одинаково толковать песню "Скрип колеса").
3. Отсутствие противоречивости внутри спека и с другими спеками (например, в поле "доставка" один программист разрешит доставку только на указанный в профиле адрес, а другой разрешит пользователю ввести любой адрес доставки; причиной тому является разная информация в спеках, полученных программистами).
4. Логическая взаимосвязь компонентов (например, достаточно вспомнить письмо собаки Шарика из "Простоквашино" родителям дяди Федора от имени сына, подписавшегося именем собаки).
5. Полнота охвата предмета (например, продюсер по незнанию не указал в спеке, что при осуществлении интернет-оплаты нужно вводить CVV код карты).
6. Соответствие нормативным актам (например, случаи с безрецептурным распространением лекарств и несоблюдением авторских прав).
7. Соответствие деловой практике (например, денежный перевод занимает 3-6 дней включительно).

Спеки имеют следующую очередность статусов:

1. Черновик (Draft) – статус во время написания спека.
2. Ожидание утверждения (Approval Pending) – статус после написания и до утверждения.
3. Утверждено (Approved или Final) – статус после утверждения.

CVS (Concurrent Version System) – система по согласованным версиям. Если мы замораживаем спек (spec freeze), она полезна для следующего:

1. С помощью CVS продюсер может сохранять версии спека и всегда вернуться к старым редакциям.
2. С помощью CVS можно «закрыть» директорию так, чтобы документы из этой директории могли читаться, но не могли редактироваться.

Неверное толкование спека (misinterpretation) – ситуация, когда программисты и (или) тестировщики понимают по-своему то, что пытался донести до них продюсер.

Тестировщики должны настаивать, чтобы спеки максимально иллюстрировались:

* макетами (mock-up),
* блок-схемами (flow chart),
* примерами (example).

Отличная идея – разработка продюсером макетов пользовательского интерфейса.

Интерфейс – это то, что видит пользователь, а алгоритм – это то, почему он видит то, что видит.

Блок-схема – process flow chart. Прекрасной идеей является включение блок-схемы в секцию тест-комплекта GLOBAL SETUP and ADDITIONAL INFO.

Блок-схемы, макеты и примеры (вместе именуемые БМП) помогают предотвратить появление багов или найти баги на уровне спека.

При обнаружении противоречий внутри спека (а БМП – это части спека!) нужно сделать рапорт о баге против продюсера, чтобы тот настроил в унисон несогласующиеся части.

Сайт в локальной сети – intranet.

### Кодирование

К документам о внутреннем дизайне кода относятся:

* документ о дизайне (архитектуре) системы (System/Architecture Design Document),
* документ о дизайне кода (Code Design Document).

Идеальный случай предусматривает наличие у программиста личной версии сайта (или playground – игровой площадки), которая включает:

* веб-сервер (web server),
* сервер с приложением (application server),
* база данных (data base).

HTML – Hyper Text Markup Language.

Первая страница при открытии сайта – homepage.

Коммуникация между браузером и сервером осуществляется посредством обмена сообщениями, основанными на HTTP – Hyper Text Transfer Protocol. Потоки таких сообщений, передающихся через интернет, называются HTTP-трафиком (HTTP traffic).

Compiler – ПО для переведения (например, C++) кода в машинный язык и создания исполняемых файлов.

Team Building Activities – мероприятия по сплочению коллектива.

Fix – фиксирование (починка, ремонт).

Code inspection – инспекции кода.

Причины багов в коде:

* некачественные и/или изменяющиеся спецификации,
* личностные качества программиста,
* отсутствие опыта,
* пренебрежение стандартами кодирования,
* сложность системы,
* баги в ПО третьих лиц,
* отсутствие юнит-тестирования,
* нереально короткие сроки для разработки.

Юнит-тестирование (unit testing) – это тестирование, которое выполняет непосредственно программист.

Рекомендации по юнит-тестированию:

1. Юнит-тесты должны планироваться в письменной форме до написания кода.
2. Требования к юнит-тестам должны быть формализованы в стандартах о юнит-тестировании.

Три основных занятия программиста:

1. Написание кода для данного релиза происходит во время стадии «Кодирование».
2. Интеграция кода для данного релиза происходит по завершении стадии «Кодирование».
3. Ремонт багов для данного релиза происходит во время стадии «Кодирование» следующего витка цикла разработки ПО (соответственно в пункте 1 программист ремонтировал баги для предыдущего релиза).

Перед началом проверки тестировщику необходимо убедиться, что:

* код заморожен (обычно релиз-инженеры посылают соответствующий e-mail),
* версия продукта на внутреннем сайте, на котором будет выполнять проверку тестировщик, является именно той версией, которую вам нужно протестировать.

Для предотвращения ситуации, когда тестировщик тестирует не ту версию ПО, необходимо:

1. Узнать у релиз-инженера, как определить версию кода, и использовать данное знание перед началом исполнения тестирования.
2. Посоветовать, чтобы внутренние веб-сайты имели логичные имена. К примеру, версия кода, переданного пользователю, должна быть на внутреннем сайте по адресу [www.old.testshop.rs](http://www.old.testshop.rs), а версия для следующего релиза – на [www.main.testshop.rs](http://www.main.testshop.rs).
3. Попросить релиз-инженеров, чтобы те создали в интранете динамически обновляемую страницу с информацией о версии и подверсии (билде) каждого внутреннего тестировочного веб-сайта.

**Синтаксический баг** (syntax bug) – самый простой в нахождении вид бага, который присущ данной стадии кодирования.

**Логический баг** (logical bug) – ошибки в логике кода, т.е. код компилируется без синтаксических ошибок, однако фактический результат исполнения этого кода не соответствует ожидаемому.

Тест-комплекты также необходимо хранить в CVS.

Преимущества хранения тест-кейсов в CVS:

* отсутствие возможности случайного удаления,
* возможность возвращения к предыдущим версиям,
* хранение файла на сервере, что дает преимущество в возможности использования его любым заинтересованным лицом, чтобы исполнить тестирование, изменить и удалить существующие или включить дополнительные тест-кейсы.

**Рассмотрение тест-кейсов** (Test-case Review) – собрание продюсера, программиста и тестировщика с целью обсуждения тест-кейсов. Полезность данного мероприятия заключается в предоставлении со стороны продюсера и программиста новых идей для тестирования и (или) корректировки допущенных неточностей.

После рассмотрения тест-кейсов рекомендуется отправить e-mail всем присутствующим на совещании перечисления всех модификаций к тест-кейсам, о которых шла речь на совещании.

### Исполнение тестирования и ремонт багов

После завершения интеграции кода тестировщики проводят тест приемки (smoke test, sanity test или confidence test), в процессе которого проверяются основные функциональности.

В случае прохождения теста приемки код замораживается и тестировщики приступают к **тестированию новых компонентов** (new feature testing), т.е. исполнению своих тест-кейсов, написанных по спекам данного релиза.

После тестирования новых функциональностей наступает очередь исполнения «старых» тест-кейсов. Данный процесс называется **регрессивным тестированием** (regression testing), которое проводится для того, чтобы удостовериться в работоспособности компонентов ПО, которые работали раньше.

После этого баги заносятся в **систему трекинга багов** (Bug Tracking System), программисты их ремонтируют, и затем тестировщики проверяют, насколько качественным был ремонт.

Acceptance or Certification Test – тест сдачи.

### Релиз

Классификация релизов:

1. Релиз (основной релиз) (major release) – стадия в цикле разработки ПО, идущая за стадией тестирования и ремонта багов, т.е. передача пользователям кода новой версии нашего ПО. Как правило, обозначается целыми числами, например, 7.0.
2. Дополнительный релиз (minor release) – ситуация, когда после основного релиза планово выпускается новая функциональность или изменяется/удаляется старая. Дополнительный релиз не связан с багами. Как правило, обозначается десятыми, например, 7.1.
3. Заплаточный релиз (patch release) – ситуация, когда после обнаружения и ремонта бага выпускается исправленный код. Как правило, обозначается сотыми, например, 7.11.

Repository – репозитарий, хранилище данных.

Билд (build) – это версия версии ПО.

Перед проверкой починки бага нужно убедиться не только в тестировании нужной версии, но и нужного билда. Номер билда, содержащего отремонтированный код, включается программистом в запись о баге в системе трекинга багов.

Первый релиз происходит следующим образом:

1. Подготовка машины у хостинг-провайдера (production server, productions или live machine – машина для пользователей).
2. Подготовка релиз-скрипта (release script) – программы, которая автоматизирует процесс релиза на машину для пользователей.
3. Исполнение релиз-скрипта:
4. релиз-скрипт запускает билд-скрипт, чтобы на тест-машине создался новый билд;
5. релиз-скрипт берет файлы этого нового билда и по протоколу FTP (File Transfer Protocol) пересылает их в машину для пользователя;
6. релиз-скрипт:

* копирует из CVS на машину для пользователя скрипты для базы данных (DB-scripts),
* запускает эти скрипты.

Скрипты для базы данных создают или модифицируют схему базы данных.

**Схема базы данных** – это совокупность виртуальных контейнеров.

**Данные** – это начинка этих виртуальных контейнеров.

Необходимость CVS для разработки ПО и релиза заключается в следующем:

* программисты будут уверены, что написанные ими версии не будут изменены или удалены другими программистами,
* возможность использования старых версий ПО в случае неработоспособности новых.

Branch – ветвь.

Полезность бранчей CVS заключается в следующем:

* релиз-инженеры смогут всегда предложить старую версию ПО, если новая оказалась некачественной;
* программисты смогут одновременно ремонтировать баги для предыдущих версий и кодировать текущую версию ПО, учитывая разделение между версиями в CVS.

У бранча есть 3 состояния:

1. Открытый, т.е. в нем можно сохранять файлы.
2. Условно открытый, т.е. в нем можно сохранять файлы, но при определенном условии. К примеру, программист должен написать номер реального бага в комментарии при сохранении файла.
3. Закрытый, т.е. файл может быть сохранен в соответствии с процедурой о неотложном ремонте багов.

На каждый баг, для которого был произведен патч-релиз, должен быть написан тест-кейс приоритета 1. Этот тест-кейс добавляется к группе тест-кейсов для регрессивного тестирования соответствующей функциональности.

Emergency Bug Fix Procedure – процедура о неотложном ремонте багов. Проводится в случае критического бага во время релиза на машину для пользователей.

В случае необходимости срочного изменения ПО, не связанного с багами, заносится запись в СТБ типа «Feature Request» – запрос о новой функциональности.

Процедура о неотложном ремонте багов должна содержать:

* приоритет багов, которые подлежат НРБ;
* список лиц, имеющих право инициировать процесс НРБ;
* последовательность действий между лицами, участвующими в НРБ.
* коммуникацию между лицами, участвующими в НРБ.

Beta release – бета-релиз.

Beta testers – бета-тестировщики.

Target group – целевая аудитория.

Альфа-тестирование – любое тестирование кода до передачи его пользователям (включая бета-пользователей). При этом юнит-тестирование является частью альфа-тестирования.

Вместо слова release может использоваться слово push – толчок.

# ***Блок 5***

# Цикл тестирования ПО

Цикл тестирования ПО состоит из трех этапов:

1. Изучение и анализ предмета тестирования.
2. Планирование тестирования.
3. Исполнение тестирования.

**Изучение и анализ предмета тестирования** начинаются перед утверждением спека и продолжаются на стадии «Кодирование».

**Планирование** тестирования происходит на стадии «Кодирование».

**Исполнение** тестирования происходит на стадии «Исполнение тестирования и ремонт багов».

## Изучение и анализ предмета тестирования

Чтобы удовлетворить потребности пользователя, необходимо:

* придумать (продюсер),
* написать (программист),
* протестировать (тестировщик),
* передать пользователям (релиз-инженер)

средства, которые удовлетворяют эти потребности. Этими средствами являются ФУНКЦИОНАЛЬНОСТИ интернет-проекта.

**Функциональность** (functionality, feature) – это средство для решения определенной задачи.

**Функциональное тестирование** (functional testing) – это проверка работы функциональностей.

Функциональности – это продукт, предоставляемый пользователям интернет-компанией.

Основные источники знания о функциональностях:

* документация,
* хомо сапиенс,
* веб-сайт.

Эксплоринг (exploring – исследование, разведка) – это изучение того, как работает сайт с точки зрения пользователя.

Интернет-компании используют эксплоринг в двух случаях:

* написан код и отсутствует документация.
* для самообучения.

Конечная цель этапа Изучение и анализ предмета тестирования:

* какие функциональности необходимо тестировать?
* как эти функциональности работают?

## Планирование тестирования

Мудрость найденных решений проявляется в двух вещах:

* кратких, простых и изящных путях для проверки функциональностей;
* компромиссе между объемом тестирования, который возможен в теории и объемом тестирования, возможным на практике.

Тест-документация – test documentation.

Tool – компьютерная программа.

## Исполнение тестирования

Исполнение тестирования заключается в практическом поиске багов в написанном коде с использованием ранее созданных тест-кейсов.

Исполнение функционального тестирования выглядит следующим образом:

* тестирование новых функциональностей (new feature testing),
* регрессивное тестирование (regression testing).

Процесс тестирования:

* после нахождения бага тестировщик вносит запись о нем в систему трэкинга багов,
* после ремонта бага программистом тестировщик проверяет:

1. действительно ли баг починен. Проверка осуществляется путем генерации ввода, который привел к выводу, не соответствующему ожидаемому результату;
2. не появились ли новые баги как нечаянное следствие изменения кода при починке. Проверка осуществляется путем тестирования функциональностей, работа которых могла быть затронута починкой.

Тестирование, исполняемое в пунктах а) и b), также называется регрессивным тестированием (bug regression testing).

Regress that bug – проведи регрессивное тестирование этого бага. В итоге нам необходимо исполнить пункты а) и b).

Объединим этапы «Изучение и анализ предмета тестирования» и «Планирование тестирования» в **подготовку к тестированию** (test preparation или test preps).

В итоге у нас имеется:

1. Подготовка к тестированию (test preparation).
2. Исполнение тестирования (test execution).

# ***Блок 6***

## Классификация видов тестирования

### По знанию внутренностей системы

1. Черный ящик (black box testing).
2. Серый ящик (grey box testing).
3. Белый ящик (white box testing).

### По объекту тестирования

1. Функциональное тестирование (functional testing).
2. Тестирование интерфейса пользователя (UI testing).
3. Тестирование локализации (localization testing).
4. Тестирование скорости и надежности (load/stress/performance testing).
5. Тестирование безопасности (security testing).
6. Тестирование опыта пользователя (usability testing).
7. Тестирование совместимости (compatibility testing).

### По субъекту тестирования

1. Альфа-тестировщик (alpha tester).
2. Бета-тестировщик (beta tester).

### По времени проведения тестирования

1. До передачи пользователю – альфа-тестирование (alpha-testing).

* тест приемки (smoke test, sanity test или confidence test);
* тестирование новых функциональностей (new feature testing);
* регрессивное тестирование (regression testing);
* тест сдачи (acceptance or certification test);

1. После передачи пользователю – бета-тестирование (beta testing).

### По критерию «позитивности» сценариев

1. Позитивное тестирование (positive testing).
2. Негативное тестирование (negative testing).

### По степени изолированности тестируемых компонентов

1. Компонентное тестирование (component testing).
2. Интеграционное тестирование (integration testing).
3. Системное (или энд-ту-энд) тестирование (system or end-to-end testing).

### По степени автоматизированности тестирования

1. Ручное тестирование (manual testing).
2. Автоматизированное тестирование (automated testing).
3. Смешанное/полуавтоматизированное тестирование (semi automated testing).

### По степени подготовки к тестированию

1. Тестирование по документации (formal/documented testing).
2. Эд хок-тестирование (ad hoc testing).

**Черный ящик** – это тестируемые части бэк-энда (код программиста, схема базы данных), составляющие невидимый пользователю виртуальный мост, соединяющий:

* фактический ввод (шаги) и
* фактический вывод (фактический результат).

Признаки подхода «черный ящик»:

1. Тестировщик не знает, как устроен виртуальный мост.
2. Идеи для тестирования идут от предполагаемых паттернов (pattern – образец, шаблон) поведения пользователей. В связи с этим подход «черный ящик» также называют **поведенческим**.

**Сценарий** (scenario) – это последовательность действий для достижения фактического результата.

Если исполнение тестирования идет по тест-кейсам, то можно сказать, что сценарий тест-кейса – это совокупность шагов тест-кейса.

**Данные для сценариев** (или просто «данные») – это конкретные значения ввода, используемые для достижения фактического результата.

Основные источники предполагаемых паттернов поведения пользователей могут быть:

* напрямую взяты из спека,
* найдены путем эксплоринга,
* получены путем применения методики черноящичного тестирования (black box testing methodology),
* подарены интуицией,
* порекомендованы программистом и продюсером,
* другими.

**Белый ящик** также называется стеклянным ящиком (glass box), открытым ящиком (clear box) или никаким ящиком (no box).

При подходе «белый ящик» тестировщик основывает идеи для тестирования на знании об устройстве и логике тестируемой части бэк-энда.

**Тестировочное покрытие** (test coverage) состоит из двух вещей:

* покрытие возможных сценариев,
* покрытие исполнения тест-кейсов.

Покрытие возможных сценариев в основном является величиной абстрактной, так как мы не знаем количество тест-кейсов, необходимых для 100% проверки ПО.

Покрытие исполнения тест-кейсов является величиной конкретной, и выражается процентным отношением исполненных тест-кейсов к общему количеству тест-кейсов.

Симбиоз использования подходов «Черный ящик» и «Белый ящик» увеличивает покрытие возможных сценариев:

* количественно, так как появляется большее количество тест-кейсов;
* качественно, так как ПО тестируется принципиально разными подходами: с точки зрения пользователя и с точки зрения внутренностей бэк-энда.

**Серый ящик** (gray/grey box) сочетает в себе элементы двух предыдущих подходов.

Существует разница между тестированием интерфейса пользователя (например, проверяется максимальное количество символов, которые можно напечатать в поле "Имя" на странице "Регистрация") и тестированием с помощью интерфейса пользователя (например, тестирование бэк-энда и задействование интерфейса для создания транзакции покупки).

Тестирование локализации связано с проверкой аспектов по адаптации сайта для пользователей из других стран (например, ввод на сайте знакомств информации о себе символами Kanji, а не английскими буквами).

Обычно целью тестирования скорости и надежности является обнаружение слабого места (bottleneck) в системе. Например, проверка скорости работы и функционирования сайта в целом при наплыве пользователей.

Киберпреступность – cyber crime.

Тестирование опыта пользователя связано с проверкой удобства использования сайта пользователем. К примеру, на сайте пиццерии в глаза пользователю сразу же бросается месторасположение ближайшего ресторана.

Интерфейс сделан с заботой об опыте пользователя (user friendly interface).

Верх страницы – header. Низ страницы – footer.

Тестирование с разными браузерами – кросс-браузер тестирование (cross-browser testing).

Тестирование с разными ОС – кросс-платформ тестирование (cross-platform testing).

Сценарий, проверяющий ситуацию, связанную с:

* потенциальной ошибкой (error) пользователя и (или)
* потенциальным дефектом (failure) в системе,

называется **негативным**.

Создание и исполнение тест-кейсов с негативными сценариями называется **негативным тестированием**.

**Позитивный** **сценарий** – сценарий, предполагающий нормальное, «правильное»:

* использование и (или)
* работу системы.

**Обращение с ошибкой (дефектом)** (error handling/failure handling) – это то, как система реагирует на ошибку/дефект.

**Сообщение об ошибке** (error message) – это информация, которая выдается пользователю в случае ошибки (сбоя).

Компонентное тестирование – тестирование на уровне логического компонента. Также это тестирование самого логического компонента.

Интеграционное тестирование – это тестирование на уровне двух и более компонентов. Также это тестирование взаимодействия двух и более компонентов.

Системное тестирование – это проверка всей системы от начала до конца.

Значения отделены друг от друга запятой – comma-delimited.

Укороченный вид – truncated.

*Тестируя интернет-проекты, приходится создавать множество пользовательских счетов, в которых используются e-mail. Чтобы не создавать массу e-mail, необходимо поговорить с админом почтового сервера вашей компании, чтобы он модифицировал настройки сервера так, чтобы к вам приходили все e-mail следующего формата:* [*rsavin+sometext@testshop.rs*](mailto:rsavin+sometext@testshop.rs)*. Таким образом, после алиса стоит + и любые легитимные знаки. К, примеру, для тестирование компонента 1 можно использовать e-mail* [*rsavin+component1\_test@testshop.rs*](mailto:rsavin+component1_test@testshop.rs)*.*

Проверяльщики линков – link checkers.

# ***Блок 7***

# Подготовка к тестированию

Подготовка к тестированию включает:

1. Написание новых тест-кейсов и (или)
2. Изменение существующих тест-кейсов и (или)
3. Удаление существующих тест-кейсов.

В классическом виде тест-кейс является комбинацией:

* идеи тест-кейса,
* сценария,
* ожидаемого результата.

Методология создания тест-кейсов – testcase design methodology.

Методы генерирования тестов – methods of test generation.

Методы отбора тестов – test selection criterion.

Методы генерирования тестов:

1. Черновик-чистовик (dirty list-white list).
2. Матричная раскладка (matrices).
3. Блок-схемы (flowchart).

Методы отбора тестов:

1. Оценка риска (risk estimate).
2. Эквивалентные классы (equivalent classes).
3. Пограничные значения (boundary values).

*Генерация черновика по тестированию автомата для продажи банок с колой.*

*Черновик:*

* Проверить, что покупателю выдается именно та банка, которую он хочет.
* А что, если покупатель нажмет на кнопку два раза?
* А что, если покупатель попробует наклонить аппарат, чтобы банки посыпались как из рога изобилия?
* Проверить, что правильно выдается сдача.
* Какая реакция на монетку иностранного государства?

**Проверить, что покупателю выдается именно та банка, которую он хочет.**

*Чистовик*

Нажать на кнопку, соответствующую желанному виду напитка. Бросить монету. Достать банку и удостовериться в ее соответствии предпочтениям.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0001 | | 1 |
| **IDEA:** Проверка автомата на выдачу напитка Pepsi  SETUP and ADDITIONAL INFO:  Наименование товара: Pepsi  Стоимость: 1 USD | | | |
| **Revision History** | | | |
| **Created on:** 16/04/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Нажать на кнопку Pepsi 2. Бросить в специальное отверстие 1 USD 3. Достать банку | | Выдача напитка Pepsi | |
| 1. Проверить надпись на банке | |

**А что, если покупатель нажмет на кнопку два раза?**

Нажать на кнопку, соответствующую желанному виду напитка два раза. Бросить монету. Проверить количество полученных банок.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0002 | | 1 |
| **IDEA:** Проверка автомата на выдачу одной банки Pepsi  SETUP and ADDITIONAL INFO:  Наименование товара: Pepsi  Стоимость: 1 USD | | | |
| **Revision History** | | | |
| **Created on:** 16/04/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Нажать на кнопку Pepsi 2. Бросить в специальное отверстие 1 USD 3. Достать банку | | Одна банка Pepsi | |
| 1. Проверить количество полученных банок | |

**А что, если покупатель попробует наклонить аппарат, чтобы банки посыпались как из рога изобилия?**

Данное тестирование находится не в нашей компетенции, так как связано с механической частью.

**Проверить, что правильно выдается сдача.**

Нажать на кнопку, соответствующую желанному виду напитка. Бросить монету номиналом, превышающим стоимость напитка. Получить банку и проверить количество полученных обратно денег.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0003 | | 1 |
| **IDEA:** Проверка автомата на выдачу правильной сдачи  SETUP and ADDITIONAL INFO:  Наименование товара: Pepsi  Стоимость: 1 USD  Наш бюджет: 2 USD | | | |
| **Revision History** | | | |
| **Created on:** 16/04/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Нажать на кнопку Pepsi 2. Бросить в специальное отверстие 2 USD 3. Достать банку | | 1 USD | |
| 1. Проверить размер сдачи | |

**Какая реакция на монетку иностранного государства?**

Нажать на кнопку, соответствующую желанному виду напитка. Бросить иностранную монету. Проверить, выдает ли автомат напиток или нет.

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0004 | | 1 |
| **IDEA:** Проверка автомата на отказ в случае использования иностранной монеты  SETUP and ADDITIONAL INFO:  Наименование товара: Pepsi  Стоимость: 1 USD | | | |
| **Revision History** | | | |
| **Created on:** 16/04/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Нажать на кнопку Pepsi 2. Бросить в специальное отверстие любую иностранную монету | | Отказ в операции | |
| 1. Проверить факт выдачи банки | |

*Составление матрицы для регистрационной формы*



Течение, процесс – flow.

**Эквивалентный класс** – это одно или больше значений ввода, к которым ПО применяет одинаковую логику.

**Пограничные значения** – это конкретные предельные значения, образующие водораздел между эквивалентными классами.

**Пограничное тестирование** (boundary testing) – это применение метода тестирования пограничных значений.

Метод тестирования пограничных значений:

* изначально тестируется нижний предел данного класса (если он имеется),
* затем тестируется верхний предел данного класса (если он имеется),
* затем тестируется любое значение внутри данного класса,
* затем тестируется верхний предел класса, предшествующего данному классу (если предшествующий класс имеется),
* затем тестируется нижний предел класса, непосредственно следующего за данным классом (если следующий класс имеется).

**Протестировать ситуацию, когда при покупке от 300 гривен предоставляется скидка в размере 5%, а при покупке от 500 гривен предоставляется скидка в размере 10%.**

|  |  |
| --- | --- |
| **Сумма, грн.** | **Дисконт, %** |
| 300-500 | 5 |
| 500 и более | 10 |

|  |  |
| --- | --- |
| Класс 1 | 0-299,99 |
| Класс 2 | 300-499,99 |
| Класс 3 | 500 и более |

|  |  |  |  |
| --- | --- | --- | --- |
| **TC ID/Priority** | CCPG0001 | | 1 |
| **IDEA:** Тестирование «Класс 2» наших эквивалентных классов.  SETUP and ADDITIONAL INFO:  Если сумма > 300 и сумма < 499,99, то дисконт = сумма/100 х 5  Проверяемые ситуации:  А. 300  Б. 499,99  В. 420,22  Г. 299,99  Д. 500  Цифровые подразделы в графе «PROCEDURE»   1. Код с выделенным жирным шрифтом показывает часть, которую мы проверяем. 2. Возможная проблема кода. 3. Пример ошибки. | | | |
| **Revision History** | | | |
| **Created on:** 26/04/2015 by В. Изотов | | Новый тест-кейс | |
| **Execution part** | | | |
| **PROCEDURE** | | **EXPECTED RESULT** | |
| 1. Если сумма **>300** и сумма <499,99, То скидка = сумма/100 х 5. 2. Ошибка в знаке равенства и/или сумме нижнего предела. 3. Знак равенства перед «300» пропущен. | | А. 5% от 300 | |
| 1. Если сумма >300 и сумма **<499.99**, То скидка = сумма/100 х 5. 2. Ошибка в знаке равенства и/или сумме верхнего предела. 3. Вместо «499,99» указано «499,00». | | Б. 5% от 499,99 | |
| 1. Если сумма **>300** и сумма **<499.99**, То скидка = сумма/100 х 5. 2. Ошибка в знаках (>) и (<). 3. Вместо «>300» указано «<300» и вместо «<499.99» указано «>499.99». | | В. 5% от 420,22 | |
| 1. Если сумма **>300 и** сумма <499.99, То скидка = сумма/100 х 5 2. Ошибка в сумме нижнего предела и/или связующей букве «и». 3. Вместо «>300» указано «>200» и вместо буквы «и» стоит буква «или». | | Г. не равно 5% от 299,99 | |
| 1. Если сумма >300 и сумма **<499,99**, То скидка = сумма/100 х 5. 2. Ошибка в сумме верхнего предела и/или связующей букве «и». 3. Вместо «<499.99» указано «599.99» и вместо буквы «и» стоит буква «или». | | Д. не равно 5% от 500 | |

# ***Блок 8***

# Исполнение тестирования

Процесс трэкинга багов (Bug Tracking Procedure) – это процесс, который начинается с занесения бага в систему трэкинга багов (Bug Tracking System).

Концептуально система трэкинга багов (СТБ) – это инфраструктура, позволяющая:

* создавать,
* хранить,
* просматривать,
* модифицировать.

информацию о багах.

Каждый баг, занесенный в СТБ, представляет собой виртуальную учетную карточку.

Статусы бага:

* открыт (open),
* закрыт (closed),
* повторно открыт (re-open).

Занести новый баг – submit new bug.

Атрибуты бага:

1. Номер бага (bug number).
2. Краткое описание (summary) – максимально информативное и сжатое описание проблемы. Если есть номер спека, то можно давать краткое описание в следующем формате: <номер спека> : <самое краткое описание>, например, 7422: *неверное значение баланса Switch после покупки*.
3. Описание и шаги для воспроизведения проблемы (description and steps to reproduce):
4. Description: Полезная информация о баге: описание, комментарии, нюансы и т.д.
5. Steps to reproduce: Конкретные шаги для воспроизведения проблемы.
6. Bug: Фактический результат.
7. Expected: Ожидаемый результат.

Звездочка (\*) – asterisk.

Жирная метка (●) – bullet.

Размер поля – maxlength.

Ниспадающее меню (например, выбор списка стран) – pull down menu.

Если неизвестно название кнопки (например, «регистрация») при написании тест-кейсов необходимо писать «отправьте форму», «submit the form» или «submit».

Приложение (например, файла с изображением) – attachment.

Фраза «Смотри приложение» должна быть в поле «Описание и шаги для воспроизведения проблемы».

Держатель бага (для кого предназначен баг, например, для программиста) – assigned to.

«Кто за что ответствен» – Who does What.

Имя передавшего баг – assigned by.

Имя того, кто должен проверить ремонт – verifier.

Где был найден баг – found on.

Версия, в которой был найден баг – version found.

Билд, в котором был найден баг – build found.

Версия с починенным кодом – version fixed.

«К данной ситуации неприменимо» – not applicable (N/A).

Билд с починенным кодом – build fixed.

Серьезность бага – severity.

**Серьезность бага** – это степень воздействия бага (magnitude of impact) на ПО, исходя из принадлежности бага к определенной технической категории.

**Серьезность отражает технический аспект бага, а приоритет – коммерческий.**

Баг, из-за которого может сорваться запланированный релиз – showstopper (пробка).

Список для оповещения – notify list.

История изменений – change history.

Проведи регрессивное тестирование – verify.

Ремонт был успешен – fix is verified.

Ремонт был неуспешен – verification failed.

Не могу воспроизвести – can't reproduce.

**Условие появления бага** – это непосредственная ситуация, воспроизведя которую мы воспроизводим баг. Например, пользователь может добавить кредитную карту с датой истечения действия в прошлом.

**Причина появления бага** – это конкретная ошибка программиста или продюсера, результатом которой является баг (например, сделана ошибка в логике кода).

Не наш баг – 3rd party bug.

Обходной путь – workaround.

Больше не применяется – no longer applicable.

Краткое изложение, сводка – summary.

Элементы веб-страницы:

* текст (неверный текст, неправильный цвет и шрифт),
* линк (сломанный линк, ведет не туда, куда нужно),
* картинка (не отображается, показан сломанный значок),
* слинкованная картинка (linked image) (болезни линков и картинок),
* однострочное текстовое поле (textbox) (максимальное количество вводимых символов может не соответствовать спеку),
* многострочное текстовое поле (text entry area) (не умещается или нет ввода значений в соответствующую колонку базы данных),
* поле пароля (passwordfield) (размер поля пароля может быть больше или меньше того, который указан в спеке),
* радиокнопка (radio button) (не нажимается, после нажатия появляется в другом поле),
* чекбокс (не ставится галочка),
* кнопка (button) (не посылается на сервер с приложением).

Фича:

* функциональность или
* характеристика (или свойство) компонента кода, интерфеса, базы данных и т.д.

Перед началом регрессивного тестирования Verifier должен удостовериться, что версия и билд на тест-машине соответствуют значениям атрибутов Version Fixed и Build Fixed для данного бага.

# ***Блок 9***

## Тестирование новых фича

Тест приемки – это, как правило, эд хок-тестирование, при котором мы проверяем, работают ли самые базовые вещи, как, например, создание нового эккаунта. Базовыми вещами могут быть, например:

* создай новый эккаунт,
* войди в систему,
* добавь книгу в корзину.

Тест-смета – test estimation.

Критерий начала/завершения – entry/exit criteria.

Тест-план – test plan.

Тест-смета включает:

* предварительную оценку времени, необходимого на подготовку к тестированию;
* предварительную оценку времени, необходимого на тестирование новых фича.

Рекомендуется увеличение тест-сметы на 10%.

Поставщик, производитель, изготовитель – vendor.

Entry criteria – это условие для начала чего-либо.

Exit criteria – это условие для завершения чего-либо.

Тест-план – это документ, обобщающий и координирующий тестирование.

Элементы тест-плана:

1. Название тест-плана, имя автора и номер версии.
2. Оглавление с разделами тест-плана.
3. Введение, в котором мы приводим информацию о сути и истории тестируемого проекта.
4. Документация с требованиями к ПО – здесь мы перечисляем имена, номера и приоритеты спеков и/или другой документации, определяющей тестируемые фича.
5. Фича, которые будут тестироваться. Перечисляем, при необходимости комментируем и назначаем приоритет.
6. Фича, которые не будут тестироваться. Перечисляем и объясняем, почему не будут тестироваться.
7. Объем тестирования – виды тестирования, которые мы будем проводить, и разъяснения к ним.
8. Тест-документация – перечисление тест-документации, которая должна быть создана для данного проекта.
9. Тест-тулы – функциональности тест-тулов, которые должны быть созданы для тестирования проекта.
10. Критерий начала/завершения:

* критерий начала подготовки к тестированию,
* критерий завершения подготовки к тестированию,
* критерий начала исполнения тестирования,
* критерий завершения исполнения тестирования.

1. Допущения – список допущений, которые мы сделали при составлении данного тест-плана и которые сделаем при тестировании.
2. Зависимости – список вещей (с пояснениями), от которых зависит та или иная часть тестирования.
3. «Железо» и ПО – список и конфигурации «железа» и ПО, которые будут использоваться при тестировании.
4. Условия приостановки/возобновления тестирования – это условия, при которых тестирование должно быть остановлено/продолжено.
5. Ответственные лица – подробный список деепричастных лиц, контактная информация и обязанности каждого из них.
6. Тренинг – тренинг, необходимый для данного проекта.
7. Расписание – сроки, имеющие отношение к тестированию данного проекта:

* дата замораживания спеков,
* дата начала подготовки к тестированию,
* дата завершения подготовки к тестированию,
* дата интеграции и замораживания кода,
* дата начала тестирования новых фича,
* дата завершения тестирования новых фича,
* дата начала регрессивного тестирования,
* дата завершения регрессивного тестирования.

1. Оценка риска – предположение о том, как и что может пойти по неправильному пути и что мы в этом случае предпримем.
2. Прочие положения – вещи, не вошедшие в тест-план, о которых следовало бы упомянуть.
3. Утверждения – это подписи лиц, которые утвердили тест-план.
4. Приложения – например, расшифровка терминов и аббревиатур, используемых в тест-плане.

## Регрессивное тестирование

Первой группой кандидатов для регрессивного тестирования у нас будут текст-комплекты, проверяющие часть ПО, к которой принадлежат новые фича.

Второй группой кандидатов для регрессивного тестирования у нас будут текст-комплекты, проверяющие старые фича, которые зависят от части ПО с новыми фича.

Факторы, которые необходимо принять при создании сметы:

* предполагаемая сложность новых фича,
* есть ли у вас опыт тестирования похожих фича,
* опыт работы на прошлых проектах с теми же продюсером и программистом,
* будет ли интеграция нашего ПО с ПО наших бизнес-партнеров – вендоров,
* нужны ли тулы для автоматизации тест-кейсов.

Условие старта для подготовки к тестированию: все спеки должны быть заморожены.

Условие завершения подготовки к тестированию: тест-кейсы и прочие подготовительные мероприятия написаны и закончены.

Условие старта для исполнения тестирования: код заморожен.

Условие завершения исполнения тестирования: тестирование новых функциональностей и регрессивное тестирование завершено, нет открытых П1 и П2 багов.