

Тест-дизайн

Что такое тест-дизайн?

Тест-дизайн - это этап процесса тестирования ПО ,на котором проектируется и создаются тестовые случаи (тест кейсы) , в соответствии с определенными критериями качества и целями тестирования (по ISTQB).

Задача тест дизайна - понять , как тестировать и что тестировать.

Цели тест дизайна

- Покрыть все требования к функциональности;
- Придумать тесты , которые обнаружат наиболее серьезные ошибки;
- Минимизировать количество тестов, необходимых для нахождения большинства серьезных ошибок.

Важное замечание : *тесты должны покрывать всю функциональность , но это не значит , что всегда надо проверять все тесты .*

Тестовое покрытие

Тестовое покрытие (Test Civerage) - это одна из метрик оценки качества тестирования , представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

Существуют несколько разных подходов:

- Покрытие требований (Requirements Coverage) - оцениваем покрытие тестами функциональных и нефункциональных требований к продукту. Для этого строим матрицу трассировки (traceability matrix).
- Покрытие кода (Code Coverage) - оцениваем покрытие исполняемого кода тестами . Для этого отслеживаем части программного обеспечения , непроверенные в процессе тестирования .
- Тестовое покрытие на базе анализа потока управления - оцениваем покрытие , основанное на определении путей выполнения кода программного модуля и создания выполняемых тест-кейсов чтобы покрыть эти пути.

Задачи тест - дизайна:

- Проанализировать требования к продукту .
- Понять , какие нас ждут риски и особенности использования продукта ;
- Написать достаточное минимальное количество тестов (т.е., которые покрывают основной функционал);
- Разделить тесты на приемочные , критические , расширенные наборы.

Тест-дизайн это мастерство тестировщика. О того как мы будем выполнять тест дизайн , зависит то насколько хорошо мы будем тестировать. Чем лучше используются методики тем лучше качество работы.

Основные методики тест-дизайна.

Методики тест дизайна - это рекомендации и советы , которые стоит использовать , когда вы разрабатываете тесты для своего приложения.

Даже на одном и том же проекте два тестировщика могут применять к одной задаче разные методики , в зависимости от того , какую они сочли более уместной .

Методики тест-дизайна включают в себя многочисленные методы .

Выбор метода проектирования тестов .

Выбор конкретного метода проектирования тестов зависит от множества факторов , включая:

- Тип системы или компонента и ее сложность;
- Нормативные документы;
- Требования пользователей или контракта;
- Уровни и типы рисков ;
- Задачи тестирования ;
- Знания и опыт тестировщиков;
- Доступные инструменты и документации;
- Ресурсы времени и бюджет ;
- Особенности жизненного цикла разработки ;
- Ожидаемое использование системы;
- Прошлый опыт использования методов проектирования тестов для компонента или системы;
- Ожидаемые типы дефектов компонента или системы.

Тестирование методом черного ящика .

Методы черного ящика (поведенческие методы или методы основанные на поведении) основываются на анализе формальных требований , спецификаций , сценариев использования, пользовательских историй или бизнес - процессов.

Эти методы применимы как для функционального , так и для нефункционального тестирования. **Методы черного ящика сосредотачиваются на связи входных данных и выходных результатов объекта тестирования, а не на его внутренней структуре .**

Примеры таких методов:

Метод эквивалентных значений .

Эквивалентные тесты — это тесты, которые приводят к одному и тому же результату.

Классы эквивалентности.

Классы эквивалентности

Информация о покупателе:



Ваше имя*

Телефон*

E-mail

	+	-
Ваше имя	Буквы [латиница, кириллица заглавные, строчные] Спецсимволы [-]	Цифры Спецсимволы [отличные от -]
Телефон	Цифры [арабские] Спецсимволы [+ - ()]	Буквы Цифры [отличные от арабских] Спецсимволы [отличные от + - ()]
E-mail	Буквы [латиница, кириллица заглавные, строчные] Цифры Спецсимволы [кроме пробела и " () , ; < > [\]] Формат [x@x.x]	Спецсимволы [пробел и " () , ; < > [\]] Формат [отличный от x@x.x]

Позитивный + корректные данные

Негативный - некорректные данные

То есть, если мы выполним два любых теста из одного класса эквивалентности — то получим один и тот же результат.

Классы эквивалентности (2/14)

Если мы ожидаем одинакового результата от выполнения двух и более тестов, эти тесты эквивалентны. Такие множества тестов называются классами эквивалентности.

Признаки эквивалентности (несколько тестов эквивалентны, если):

- Она направлены на поиск одной и той же ошибки.
- Если один из тестов обнаруживает ошибку, другие её тоже, скорее всего, обнаружат.
- Если один из тестов НЕ обнаруживает ошибку, другие её тоже, скорее всего, НЕ обнаружат.
- Тесты используют одни и те же наборы входных данных.
- Для выполнения тестов мы совершаем одни и те же операции.
- Тесты генерируют одинаковые выходные данные или приводят приложение в одно и то же состояние.
- Все тесты приводят к срабатыванию одного и того же блока обработки ошибок («error handling block»).
- Ни один из тестов не приводит к срабатыванию блока обработки ошибок («error handling block»).

(С) 2008, ЦОТ «Белхард», авторы: Святослав Куликов, Ольга Смолякова

Классы эквивалентности (4/14)

Необходимо проверить, как работает поле, в которое можно ввести целое число в диапазоне от 1 до 99.

Классы эквивалентности здесь:

- Любое целое в диапазоне от 1 до 99. Как правило, это будет середина числового отрезка. Позитивный тест.
- Любое число меньше 1. Негативный тест.
- Любое число больше 99. Негативный тест.
- Дробь и «не число» (буквы, спецсимволы). Негативный тест.

Тесты, которые мы выполним:

- Ввести 1, 99, 50
- Ввести 0
- Ввести 100
- Ввести 50.5, букву, спецсимвол: ~`!"@'#\$;%:^^&*(){}.,V+=- _

(С) 2008, ЦОТ «Белхард», авторы: Святослав Куликов, Ольга Смолякова

Эквивалентное разбиение делит данные на группы (классы эквивалентности) , которые обрабатываются схожим образом . Области эквивалентности могут быть позитивными и негативными.

Разделение на валидность и не валидность подразумевает ваш личный опыт и исходя из нормативной документации.

Граничные значения

Метод анализа граничных значений - продолжение метода эквивалентного разбиения , **но может быть применим , только если классы состоят из упорядоченных числовых значений.**

Границы эквивалентного класса - его максимальное и минимальное значение. Чтобы применить эту технику , надо взять граничное значение , сразу до границы и после границы.

Пример: В случае формы из предыдущего примера границами будут 1 и 99. Значит проверяем значения 0 1 2 и 98 99 100.

Комментарии из ДЗ:

С точки зрения тест-дизайна некорректно писать внутри тест кейса в описании шагов : «ввести более 64 символов». Применяя анализ граничных значений вводится конкретное количество символов. Так же данная техника подразумевает чуть больше проверок.

Вместо сценария "граничные значения..." - эти проверки можно раскидать по сценариям ввода валидных и невалидных значений. Таким образом получится чуть упростить, а сценарии останутся по прежнему понятными и простыми в тестировании=)

В числовых полях:поле телефон,паспорт,календарь- нет диапазона, поэтому не надо писать проверку на "нижнюю границу". Анализ граничных значений подобным образом применяется только к полям ФИО.

Тестирование критического пути

Тестирование критического пути (critical path test) - исследование функциональности , которую используют наши обычные пользователи для своих обычных задач.

Тестирование того , чем чаще пользуются наши пользователи.

Т.е. мы проверяем наиболее важные функции и их функциональность за которой приходят наши пользователи.

Пример:Если это онлайн магазин то должны работать функции добавления товара в корзину, и оплата товара

Критический путь рассматривает приложение полностью на сценарий использования ,наиболее важные функции , самые главные и приоритетные функции , без чего приложение не будет полезным .(доп функции не являются критическими-справки информации характеристики).

Прочие методы

- Тестирование на основе таблиц принятия решений (таблица альтернатив);
- Таблица переходов (или таблица состояний и переходов);
- Комбинаторные методы;
- Тестирование с помощью вариантов использования (Use Case).

Метод белого ящика

Методы белого ящика (структурные методы или методы , основанные на структуре)(есть доступ к коду ,архитектуре ,базе данных) основываются на анализе архитектуры , детального проектирования ,внутренней структуры или кода компонента либо системы. В отличие от методов черного ящика методы белого ящика сосредотачиваются на структуре и обработке внутри объекта тестирования.

Более актуальны для опытных ручных тестировщиков и автоматизаторов, но важно о них знать.

Белый ящик :Тестирование и покрытие операторов и условий.

Проверка методом тестирования операторов направлена на проверку исполняемых операторов в коде .Фактически , суть в том что в тесте должен быть задействован каждый оператор кода хотя бы раз.

Тестирование условий направлено на проверку логических условий в коде, а также кода, выполняемого в зависимости от исхода условия.

(оператор-это элемент языка ,задающий полное описание действия,которое необходимо выполнить.Каждый оператор представляет собой законченную фразу языка программирования и определяет некоторый вполне законченный этап обработки данных. Пример: $A=A+1$)

На основе опыта (эвристики)

Методы , основанные на опыте , используют опыт разработчиков , тестировщиков и пользователей для проектирования , реализации и выполнения тестов. Их часто совмещают с методами черного и белого ящиков.

Некоторые из них:

Метод предположения об ошибках

Чем больше опыт тем лучше тестируем.

Предположение об ошибках основывается на знаниях тестировщика включающего:

- Историю работы приложения в прошлом
- Наиболее вероятные типы дефектов , допускаемых при разработке;
- Типы дефектов , которые были обнаружены в схожих приложениях;
- Данные технического задания.

Например , в ТЗ написано что “Когда с сервера приходит картинка , мы ее показываем”. Мы в таком случае проверяем , а что будет , если она не придет или придет не картинка а видео.

Реалистичный пример: разработчики сделали сборку для релиза и уверены, что раз там исправили два маленьких бага, то они ничего серьезного не сломали. Но опытный тестировщик знает, что обычно в половине случаев все ломается после сборки, и все равно все будет проверять очень внимательно.

Метод исследовательского тестирования

Во время исследовательского тестирования мы решаем , что проверяем прямо в процессе тестирования и изучении программы. Это оптика , которую можно применять к любому виду тестирования и методу тест-дизайна.

Это вариант особенно подходит в условиях нехватки документации или при сжатых сроках на тестирование.

Пример: Вышел новый закон о пользовательских данных и из за этого надо срочно добавить новую галочку в регистрации - в этом случае мы будем тестировать методом исследовательского тестирования.

Тестирование на основе чек-листов

При тестировании по чек листам тестировщик проектирует , реализует и выполняет тесты ,покрывающие тестовые условия , указанные в чек -листе.

В качестве составной части анализа тестировщики могут создавать новые чек-листы , расширять их либо использовать готовые чек-листы , не меняя их.

Такие списки могут быть построены на опыте , на исторических данных об ошибках , на информации о приоритетах для пользователей и понимании , как и почему происходят отказы в программе.

Пример чек-листа: реальный чек-лист к задаче, в которой делали новые пейволлы (страница в приложении, на которой пользователю предлагают купить подписку):

	ios уже посмотрели	ios ок багов нет	android уже посмотрели	andr. багов нет	web посмотрели
аналитика	+	+			+
переводы зашитые в клиент					
переводы приходящие с сервера					
пейвол на онбординге	+	+	+	-	-
пейвол на входе	+	+	+	+	-
пейвол в книге	+	+	+	+ -	начали смотреть
пейвол со всеми подписками / страница со всеми подписками (веб)					+
обновление с предыдущей версии					-
партнерские пейволлы	+	+			
новые юзеры	+	+	+	+	+
старые без подписки					+
старые с обычной на книги					
старые с премиумом					
старые с аудио					
старые с комбинациями					
старые на более старых версиях					-
динамические лейблы					
соответствие макетам по дизайну					
цены верные					
планы ок					
все виды оплат как ранее, работают					

Парметры принятия продукта в тестировании.

Альфа-тестирование и бета-тестирование

Альфа-тестирование и бета-тестирование обычно используют разработчики уже готового продукта, когда хотят получить обратную связь от потенциальных или существующих пользователей.

Цели альфа- и бета-тестирования:

- Получить уверенность в том, что система работает до ее запуска;
- Обнаружить баги, связанные с условиями и средой эксплуатации;
- Привлечь внимание пользователей, заинтересованных в эксклюзивности (например, в играх);
- Проверить необходимость некоторой функциональности в целом.

Затронем тему, которую часто не упоминают: когда мы можем начать тестирование продукта?

Говорим мы о новом функционале или о исправленных багах — важно контролировать, чтобы у нас были все доступные данные по нашей задаче. Это нужно для того, чтобы в полной мере понимать, какие методики нам применять и что именно от нас нужно в данный момент.

Пример того, что стоит знать по задаче:

- Подробное описание того, как должна работать функциональность;
- Описание окружения, на котором мы можем проверить сейчас и на котором будет работать потом;
- Связанные задачи, если они есть;
- Тестовые данные, если они нужны, например, учетки или некие данные.

При всем многообразии вариантов тестирования, важно помнить одно — наша задача, как специалиста, не найти баги, а проверить, что все действительно работает, как задумано. И почаще напоминать об этом разработке, конечно ;)

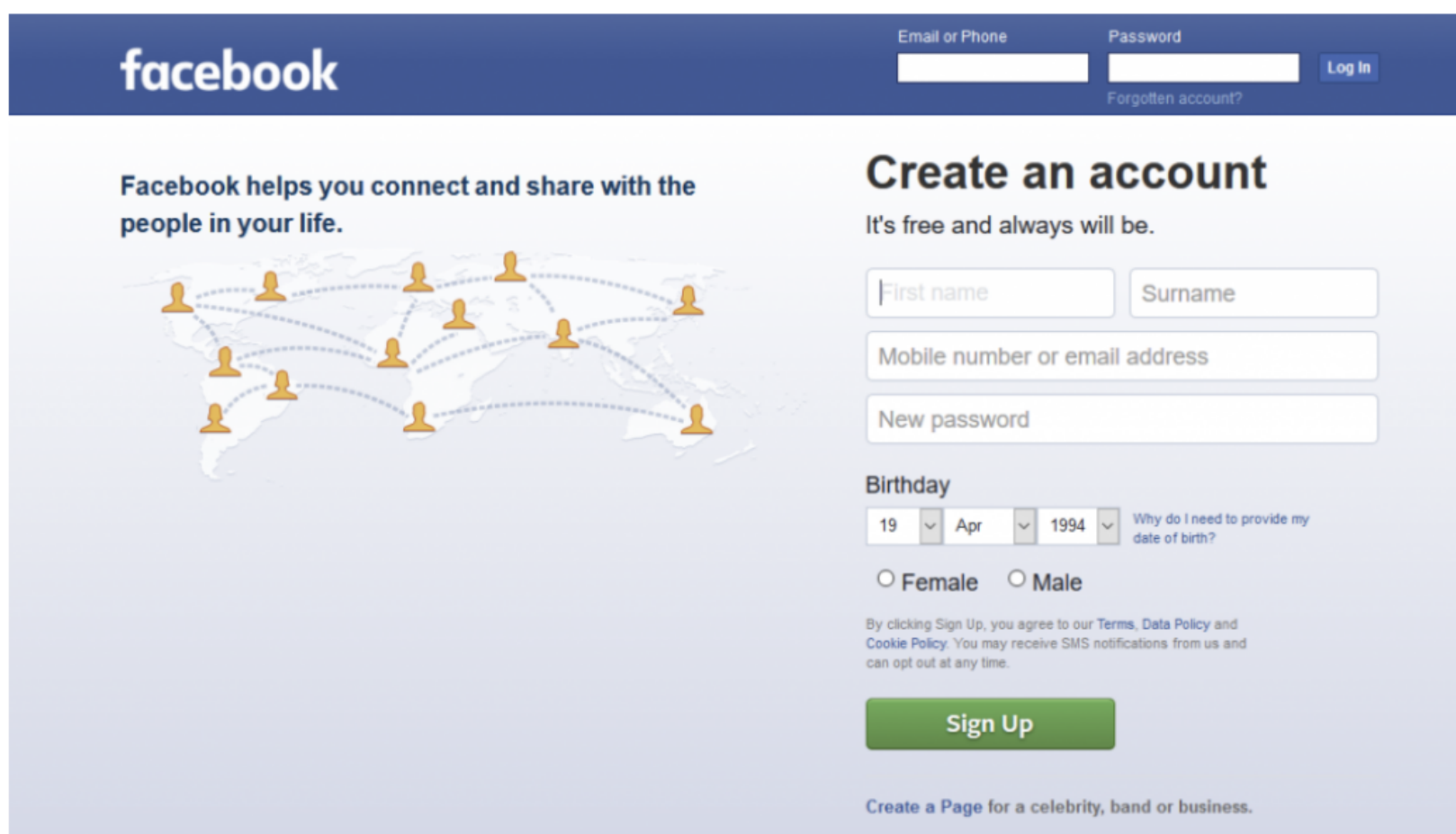
Помимо этого, вопрос, который чаще всего поможет понять, в каком направлении стоит двигаться — это **«А что, если?»**

Например, если написано, что в поле надо ввести возраст, нам нужно подумать — а что, если ввести пустое значение? Или букву? Или иероглиф? И так далее.

РАЗЛИЧИЯ МЕЖДУ ВЕРИФИКАЦИЕЙ И ВАЛИДАЦИЕЙ

Верификация	Валидация
По факту отвечает на вопрос, правильно ли создается и тестируется ПО и все ли требования учитываются при этом.	Отвечает на вопрос, создается ли продукт правильно с точки зрения ожиданий клиента.
В процессе верификации убеждаемся в том, что весь созданный функционал приложения работает корректно и логически верно.	При процессе валидации убеждаемся в том, что продукт полностью соответствует поведению, которое от него ожидается и то, что клиент знает о наличии подобного функционала.
В структуру верификации входят такие компоненты, как сверка завалидированным требованиям, технической документации и корректное выполнения программного кода на любом этапе создания и тестирования ПО.	Валидация, по своей сути, в большей степени включает в себя общую оценку ПО и может основываться исключительно на субъективном мнении касательно правильности работы приложения или его компонентов.

Теперь рассмотрим пример на основе формы входа/регистрации в рамках популярной социальной сети Facebook.

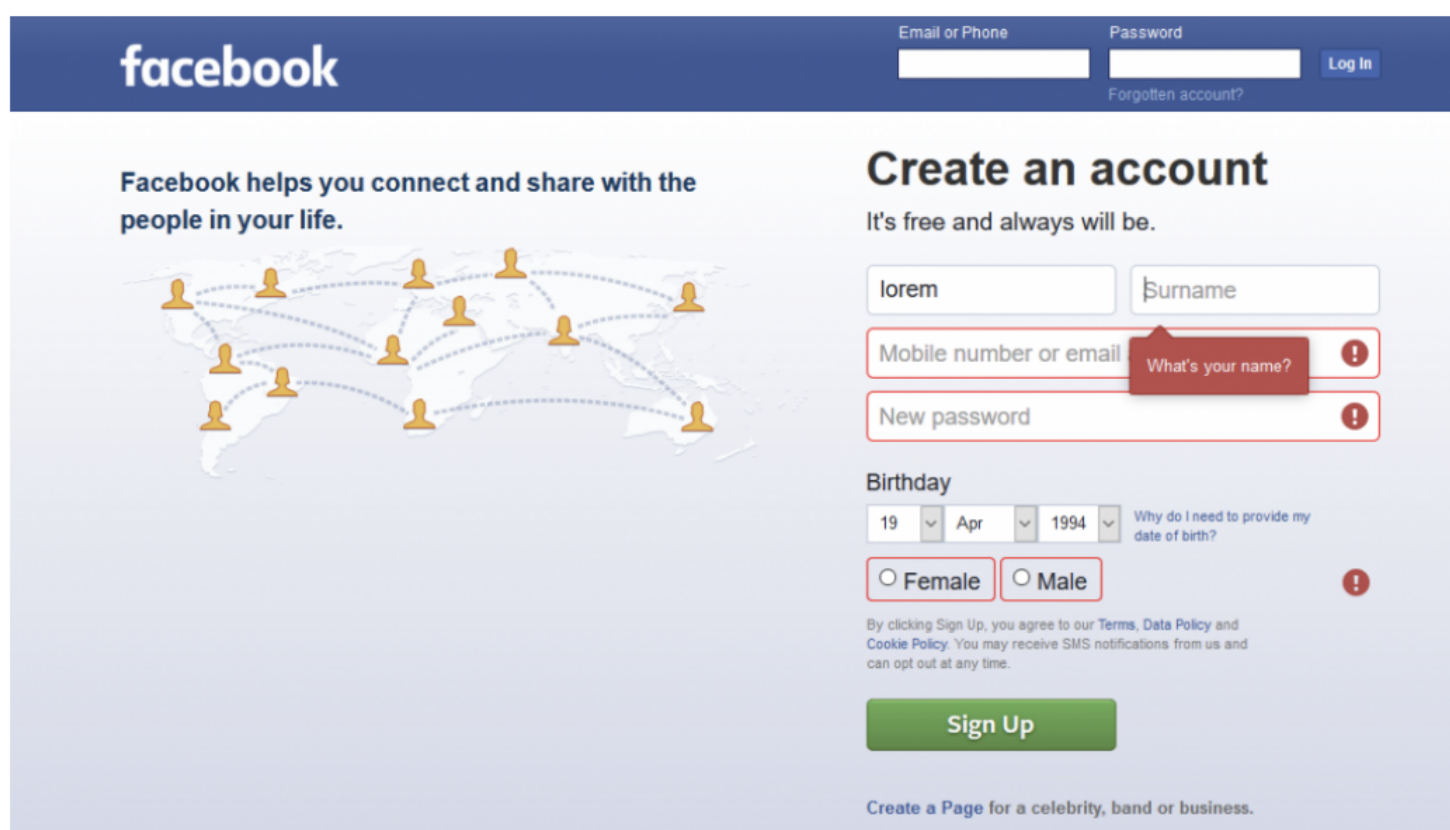


The screenshot shows the Facebook registration page. At the top, there is a blue header with the Facebook logo on the left and login fields (Email or Phone, Password, Log In) on the right. Below the header, the page is divided into two main sections. On the left, there is a graphic with the text "Facebook helps you connect and share with the people in your life." and an illustration of a world map with people icons connected by lines. On the right, there is a "Create an account" section with the text "It's free and always will be." Below this, there are input fields for "First name", "Surname", "Mobile number or email address", and "New password". There is also a "Birthday" section with dropdown menus for day, month, and year, and radio buttons for "Female" and "Male". A "Sign Up" button is at the bottom of the form. A link "Create a Page for a celebrity, band or business." is at the very bottom.

Регистрация на Facebook

Допустим, у нас есть спецификация и наличие группы полей на целевой странице ей всецело соответствует. Исходя из этого факта, верификация уже прошла успешно.

Дальше нашей задачей является проверка валидации. Чтобы долго не затягивать этот процесс, изначально введем неверную информацию в поля ввода данных.



The screenshot shows the Facebook registration page with validation errors. The "First name" field contains "lorem" and the "Surname" field contains "Surname". The "Mobile number or email" field has a red border and a red exclamation mark icon. The "New password" field has a red border and a red exclamation mark icon. The "Birthday" section has a red border and a red exclamation mark icon. A red tooltip message "What's your name?" is pointing to the "Surname" field. The "Sign Up" button is at the bottom of the form. A link "Create a Page for a celebrity, band or business." is at the very bottom.

Неверная информация при регистрации на Facebook

Как мы видим на скриншоте, введенная информация не прошла валидацию и система не пустила нас дальше, что и нужно было протестировать.

- При валидации тестируется полная работоспособность отмеченной функциональности.
- При верификации проверяется наличие в продукте этой логики (параметров взаимодействия компонентов).

Чтобы коллегам было приятнее работать с тест-кейсами, лучше делать их описание обезличенным — "Выполнить, загрузить"...

