Topic 6. Test Design Technics

Девиз сегодняшней лекции:

напишите код, а баги я в нем найду!

Содержание

1. Тест дизайн.
2. Тестовое покрытие.
3. Техники дест дизайна

Тест дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

План работы над тест дизайном

- ❑ анализ имеющихся проектных артефактов: документация (спецификации, требования, планы) и т.д.
- ❑ написание спецификации по тест дизайну
- ❑ проектирование и создание тест кейсов.

Роли, ответственные за тест дизайн
Тест аналитик - определяет "**ЧТО тестировать?**"
Тест дизайнер - определяет "**КАК тестировать?**"
Попросту говоря, задача тест аналитиков и дизайнеров сводится к тому, чтобы используя различные стратегии и техники тестдизайна, создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого ПО. Однако, на большинстве проектов эти роли не выделяется, а доверяется обычным тестировщикам, что не всегда положительно сказывается на качестве тестов, тестировании и, как из этого следует, на качестве ПО (конечного продукта).

Тестовое Покрытие - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода. Если рассматривать тестирование как "проверку соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов", то именно этот конечный набор тестов и будет определять тестовое покрытие:

Чем выше требуемый уровень тестового покрытия, тем больше тестов будет выбрано, для проверки тестируемых требований или исполняемого кода.

Для разработки набора тестов, обеспечивающего более менее высокий уровень покрытия можно использовать специальные техники тест дизайна.

Существуют следущие подходы к оценке и измерению тестового покрытия:

Покрытие требований (Requirements Coverage)- оценка покрытия тестами функциональных и нефункциональных требований к продукту путем построения матриц трассировки (traceability matrix).

Покрытие кода (Code Coverage)- оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

Тестовое покрытие на базе анализа потока управления - оценка покрытия основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Различия:
Метод покрытия требований сосредоточен на проверке соответствия набора проводимых тестов требованиям к продукту, в то время как анализ покрытия кода - на полноте проверки тестами, разработанной части продукта (исходного кода), а анализ потока управления - на прохождении путей в графе или модели выполнения тестируемых функций (Control Flow Graph).
Ограничения:
Метод оценки покрытия кода не выявит нереализованные требования, так как работает не с конечным продуктом, а с существующим исходным кодом.
Метод покрытия требований может оставить непроверенными некоторые участки кода, потому что не учитывает конечную реализацию.

Покрытие требований (Requirements Coverage)

Расчет тестового покрытия относительно требований проводится по формуле:

Tcov = (Lcov/Ltotal) * 100% где:

Tcov - тестовое покрытие
Lcov - количество требований, проверяемых тест кейсами
Ltotal - общее количество требований

Для измерения покрытия требований, необходимо проанализировать требования к продукту и разбить их на пункты. Опционально каждый пункт связывается с тест кейсами, проверяющими его. Совокупность этих связей - и является **матрицей трассировки.** Проследив связи, можно понять какие именно требования проверяет тестовый случай. Тесты не связанные с требованиями не имеют смысла. Требования, не связанные с тестами - это **"белые пятна",** т.е. выполнив все созданные тест кейсы, нельзя дать ответ реализовано данное требование в продукте или нет.

Для оптимизации тестового покрытия наилучшим способом будет использование стандартных техник тест дизайна.

Покрытие кода (Code Coverage)

Расчет тестового покрытия относительно исполняемого кода программного обеспечения проводится по формуле:

Tcov = (Ltc/Lcode) * 100% где:

Tcov - тестовое покрытие

Ltc - кол-ва строк кода, покрытых тестами

Lcode - общее кол-во строк кода.

В настоящее время существует инструментарий (например:Clover), позволяющий проанализировать в какие строки были вхождения во время проведения тестирования, благодаря чему можно значительно увеличить покрытие, добавив новые тесты для конкретных случаев, а также избавиться от дублирующих тестов. Проведение такого анализа кода и последующая оптимизация покрытия достаточно легко реализуется в рамках тестирования белого ящика (white-box testing) при модульном, интеграционном и системном тестировании; при тестировании же черного ящика (black-box testing) задача становится довольно дорогостоящей, так как требует много времени и ресурсов на установку, конфигурацию и анализ результатов работы, как со стороны тестировщиков, так и разработчиков.

Тестирование потоков управления (Control Flow Testing) - это одна из техник тестирования белого ящика, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.
Фундаментом для тестирования потоков управления является построение графов потоков управления (Control Flow Graph), основными блоками которых являются:
блок процесса - одна точка входа и одна точка выхода
точка альтернативы - одна точка входа, две и более точки выхода
точка соединения - две и более точек входа, одна точка выхода

Техники дест дизайна (Test Design Technics)

Многие люди тестируют и пишут тесткейсы, но не многие пользуются специальными **техниками тест дизайна**. Постепенно, набираясь опыта они осознают, что постоянно делают одну и ту же работу, поддающуюся конкретным правилам. И тогда они находят, что все эти правила уже описаны.

Давайте посмотрим описание наиболее распространенных техник тест дизайна:

Эквивалентное Разделение (**Equivalence Partitioning - EP**). Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала - 0.
Эквивалентный класс — это одно или больше значений ввода, к которым ПО применяет одинаковую логику.
Анализ Граничных Значений (**Boundary Value Analysis - BVA**). Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ Граничный значений может быть применен к полям, записям, файлам, или к любого рода сущностям имеющим ограничения.
Пограничным тестированием (boundarytesting) называется применение метода тестирования пограничных значений.

Для каждого эквивалентного класса может быть лишь **один из трех вариантов**:

- ❑ Есть только нижний предел
- ❑ Есть нижний и верхний пределы
- ❑ Есть только верхний предел

- ✔ Сначала 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, непосредственно предшествующего данному классу (если предшествую щий класс имеется).
- ✔ Затем тестируется нижний предел класса, непосредствен но следующего за данным классом (если следующий класс имеется)

Причина / Следствие (**Cause/Effect - CE**). Это, как правило, ввод комбинаций условий (причин), для получения ответа от системы (Следствие). Например, вы проверяете возможность добавлять клиента, используя определенную экранную форму. Для этого вам необходимо будет ввести несколько полей, таких как "Имя", "Адрес", "Номер Телефона" а затем, нажать кнопку "Добавить" - эта "Причина". После нажатия кнопки "Добавить", система добавляет клиента в БД и показывает его номер на экране - это "Следствие".

Предугадывание ошибки (**Error Guessing - EG**). Это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы "предугадать" при каких входных условиях система может выдать ошибку. Например, спецификация говорит: "пользователь должен ввести код". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание

Исчерпывающее тестирование (**Exhaustive Testing - ET**) - это крайний случай. В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным, из-за огромного количества входных значений.

Таблицы принятия решений

		Правила (rules)							
Условия	Значения	1	2	3	4	5	6	7	8
Условие1	T,F	T	T	T	T	F	F	F	F
Условие2	T,F	T	T	F	F	T	T	F	F
Условие3	T,F	T	F	T	F	T	F	T	F
Действия									
Действие1		X	X			X			
Действие2		X					X		
Действие3			X			X	X		
Действие4				X	X			X	X

Условия – входящие данные

Действия – исходящие данные (ожидаемый результат)

Правила – тест-кейсы

Таблицы принятия решений

Шаги построения таблицы

1. Определить/записать все условия
2. Посчитать количество возможных комбинаций условий N = n1*n2*…nm
3. Заполнить комбинации
4. Записать действия
5. Убрать лишние комбинации.

Автомобиль находится перед светофором. Определить его дальнейшие действия.

Условия:
Горит ли красный? Y, N
Горит ли желтый? Y, N
Горит ли зеленый? Y, N

Количество комбинаций:
N = 2*2*2 = 8

Таблицы принятия решений

Условия	Значения	1	2	3	4	5	6	7	8
Горит красный?	Y, N	Y	Y	Y	Y	N	N	N	N
Горит желтый?	Y, N	Y	Y	N	N	Y	Y	N	N
Горит зеленый?	Y, N	Y	N	Y	N	Y	N	Y	N

Действия								
Ехать							X	
Стоять		X		X				
Готовиться		X						
Специальное действие	X		X		X	X		X

Таблицы принятия решений

Условия	Значения	1	2	3	4	5	6	7	8	9	10	11	12
Горит красный?	Y, N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Горит желтый?	Y, N, Blinking	Y	Y	N	N	B	B	Y	Y	N	N	B	B
Горит зеленый?	Y, N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

Действия													
Ехать							X		X			
Стоять		X		X								
Готовиться		X					X					
Спец. действие	X		X		X	X		X		X	X	
Нерегулируется												X

Таблица принятия решений

	A	B	C	D	E	F	G	H	I	J	K
1	**These Decision Table helps to test access to the package item depending on set of conditions**										
2											
3	**How to read test cases**	**Example 3 column**									
4	1. create package with manifest 2. set for the same item Role"D" as deny and Everyone Role as allow 3. add to Role"D" user and user group 4. save package in the system 5. run package by user 6. run package by user from user group 7. run package by user who doesn't belong to user group and is not user from step 5	Expected Result: 5. denied access for user 6. denied access for user from user group 7. allow access for other user									
5											
6	**According to example check all test cases. Go to the table, take set of conditions and do case**										
7											
8	item - It is any item in the package.										
9					Test cases						
10	**Conditions**	**Variants of conditions**	1	2	3	4	5	6	7	8	9
11	item	Deny Role (D), EveryoneDeny (E), No Role (N)	D	D	D	D	E	E	E	N	N
12	item	Allow Role (A), EveryoneAllow (E), No Role (N)	A	A	E	E	N	A	E	A	E
13	user in the Role	Y, N	Y	N	Y	N	-	Y	-	Y	-
14	group in the Role	Y, N	Y	N	Y	N	-	Y	-	Y	-
15											
16	**Actions**	**Variants of Actions**									
17	Access	User (U), Member of Group (G)								U, G	
18	Denied	User (U), Member of Group (G), Everyone(E)	U, G	E	U, G		x				
19	Access to all other users/groups				x						
20	Denied to all other users/groups							x		x	
21	Full Access					x					x
22	Denied for All							x			
23	impossible								x		
24	**Pass/Fail**		Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pas
25	Jira issue							AG-24032	AG-23845		
26	Resolution							ByDesign	Fixed		
27											

	A	B	C	D	E	F	G	H	I	J	K
1	**These Decision Table helps to test access to the package item depending on set of conditions**										
2											
3	**How to read test cases**	**Example 3 column**									
4	1. create package with manifest 2. set for the same item Role"D" as deny and Everyone Role as allow 3. add to Role"D" user and user group 4. save package in the system 5. run package by user 6. run package by user from user group 7. run package by user who doesn't belong to user group and is not user from step 5	Expected Result: 5. denied access for user 6. denied access for user from user group 7. allow access for other user									
5											
6	**According to example check all test cases. Go to the table, take set of conditions and do case**										
7											
8	item - It is any item in the package.										
9					**Test cases**						
10	**Conditions**	**Variants of conditions**	1	2	3	4	5	6	7	8	9
11	item	Deny Role (D), EveryoneDeny (E), No Role (N)	D	D	D	D	E	E	E	N	N
12	item	Allow Role (A), EveryoneAllow (E), No Role (N)	A	A	E	E	N	A	E	A	E
13	user in the Role	Y, N	Y	N	Y	N	-	Y	-	Y	-
14	group in the Role	Y, N	Y	N	Y	N	-	Y	-	Y	-
15											
16	**Actions**	**Variants of Actions**									
17	Access	User (U), Member of Group (G)								U, G	
18	Denied	User (U), Member of Group (G), Everyone(E)	U, G	E	U, G		x				
19	Access to all other users/groups				x						
20	Denied to all other users/groups							x		x	
21	Full Access					x					x
22	Denied for All							x			
24	**Pass/Fail**		Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pas
25	Jira issue							AG-24032	AG-23845		
26	Resolution							ByDesign	Fixed		

Статус тест кейса

	A	B	C	D	E	F	G	H	I	J	K
1	**These Decision Table helps to test access to the package item depending on set of conditions**										
3	**How to read test cases**	**Example 3 column**									
4	1. create package with manifest 2. set for the same item Role"D" as deny and Everyone Role as allow 3. add to Role"D" user and user group 4. save package in the system 5. run package by user 6. run package by user from user group 7. run package by user who doesn't belong to user group and is not user from step 5	Expected Result: 5. denied access for user 6. denied access for user from user group 7. allow access for other user									
6	**According to example check all test cases. Go to the table, take set of conditions and do case**										
8	item - It is any item in the package.										
9											
10	**Conditions**	**Variants of conditions**	1		2	3	4	5	6	7	8	9
11	item	Deny Role (D), EveryoneDeny (E), No Role (N)	D	D		D		E	E	E	N	N
12	item	Allow Role (A), EveryoneAllow (E), No Role (N)	A	A		E		N	A	E	A	E
13	user in the Role	Y, N	Y	N		Y		-	Y	-	Y	-
14	group in the Role	Y, N	Y	N		Y		-	Y	-	Y	-
15												
16	**Actions**	**Variants of Actions**										
17	Access	User (U), Member of Group (G)									U, G	
18	Denied	User (U), Member of Group (G), Everyone(E)	U, G	E		U, G		x				
19	Access to all other users/groups					x						
20	Denied to all other users/groups								x		x	
21	Full Access						x					x
22	Denied for All								x			
23	impossible									x		
24	**Pass/Fail**		Pass	Pass		Pass	Pass	Pass	Pass	Pass	Pass	Pas
25	Jira issue								AG-24032	AG-23845		
26	Resolution								ByDesign	Fixed		
27												

Один тест кейс расписать подробно с STR

Литература:

1. http://www.protesting.ru/
2. http://ru.qahelp.net/
3. http://habrahabr.ru/
4. The Scrum Master Training Manual, v. 1.2., By Nader K. Rad, Frank Turley, Copyright © 2013 Management Plaza.
5. «Тестирование Дот Ком или пособие по жесткому обращению с багами в интернет-стартапах» Р. Савин.
6. SQADays-14, Елена Сташенко.