Анализ граничных величин

Все тестировщики как минимум наслышаны о таких техниках тест-дизайна, **как классы эквивалентности и анализ граничных значений.** Казалось бы, что может быть проще: выделить классы, взять по одному значению в каждом, проверить границы классов и значения слева и справа от границ. Но всегда ли дела обстоят настолько просто? Как быть, если после разбиения на классы оказывается, что с границами, в общем-то, проблема — их нельзя определить, поскольку данные невозможно упорядочить? Что если тестируемые параметры связаны между собой некоей логикой и зависят друг от друга? Сколько тестов достаточно? Ниже будут рассмотрены возможности двух основных техник тест-дизайна, превышающие те, что заложены в их непосредственном определении.

Область определения — математический термин — совокупность всех возможных значений переменной. Тестирование областей определения рассматривает программу как функцию многих переменных, каждая из которых принимает конечное множество значений. Каждое такое множество можно разбить как минимум на два класса эквивалентности — валидные и невалидные значения.

Тестирование областей определения предполагает три шага:

1. выделение подобластей для каждого параметра, все элементы которых предположительно приводят к одинаковому поведению программы (для сокращения количества тестов);
2. выбор конкретных значений для тестирования внутри каждого класса (в т.ч. для выявления ошибок, связанных с тем, что область определения задана неверно);
3. сочетание этих значений (для увеличения тестового покрытия и выявления ошибок, зависящих от взаимодействия нескольких параметров).

Опытные (и не очень) тестировщики сейчас скептически заметят, что задачи выше решаются тремя техниками тест-дизайна — разбиением на классы эквивалентности, анализом граничных значений и попарным перебором. Это действительно так, потому дальше речь пойдет о некоторых особенностях применения этих техник, которые помогут обнаружить больше ошибок и/или сократить время тестирования, при этом максимально сохранив уровень тестового покрытия, а значит, и уверенность в качестве программы.

Классы эквивалентности

Несколько простых правил.

1. Если область определения параметра — диапазон, то имеет смысл выделение трех классов эквивалентности: слева от диапазона (невалидные значения), сам диапазон (валидные значения) и справа от диапазона (снова невалидные). При выделении классов нужно использовать включающие границы с целью однозначности и точности: одно и то же значение не может относиться к двум классам одновременно.
2. Если область определения — набор неупорядоченных данных, то всегда можно выделить как минимум два класса — валидные и невалидные значения. Полученное разбиение можно «дробить» дальше. Например, множество латинских букв можно разбить на два подмножества: латиница в верхнем и нижнем регистре соответственно.

В примере выше используется очевидный способ дробления на подклассы, но он не единственный. Такое разбиение не всегда целесообразно в том смысле, что с его помощью не так уж часто находятся ошибки. Вот некоторые другие приемы:

- по частоте использования конечными пользователями (например, для параметров типа логина и пароля может иметь смысл выделение в отдельный класс символов «qwertyQWERTY1234567980»);
- случайные равные по размеру подклассы (обеспечение условного тестового покрытия, если нет никаких других логических способов выполнить разбиение на подклассы).

Различают линейные (упорядоченные) и нелинейные (неупорядоченные) классы эквивалентности. Очевидно, к последним невозможно применить анализ граничных значений, т.е. нет логического способа выделить элементы, с большей вероятностью приводящие к ошибке. Примером такого класса может быть множество специальных символов, которые можно ввести с клавиатуры. Для дробления такого класса может пригодиться второй прием. Входным параметром для его применения будет количество подклассов, которые планируется использовать в тестировании: из каждого будет взято по одному значению.

Типичные ошибки этого этапа тестирования областей определения: слишком много или слишком мало классов, классы выделены неправильно (по

отношению к функциональности программы).

Выбор значений

После того, как завершено разбиение на классы эквивалентности, необходимо выбрать значения из каждого класса, которые будут использоваться в тестах. Анализ граничных значений — только один из способов, причем подходящий только для линейных классов. Что же предпринять в других случаях?

1. **Случайный выбор.** Крайне желательно, чтобы при каждом следующем выполнении теста наугад выбиралось какое-то другое значение из класса. В этом случае можно применять случайный выбор с возвратом (значение, выбранное ранее, может быть выбрано повторно с той же вероятностью) или без возврата (значение, выбранное ранее, не может быть больше выбрано, тем самым вероятность выбора оставшихся в классе значений увеличивается).
2. **Пропорциональное разбиение.** Есть разные алгоритмы, основная цель которых — уменьшить риск неправильного разбиения на классы эквивалентности. Для этого можно брать несколько значений из каждого класса (число увеличивается с увеличением полезности той или иной функции программы). Другой способ: взять из каждого класса не фиксированное количество значений, а фиксированную часть класса. Таким образом, для классов, содержащих больше элементов, получится больше тестов.
3. **Анализ граничных значений.** Следует помнить, что основная идея этой техники — выделение значений, приводящих к ошибкам с большей вероятностью, чем другие. Эта техника не ограничивается непосредственно элементами управления на экране программы. Кроме числовых границ диапазонов стоит помнить о временных границах (например, срок бесплатного пользования программой), границах циклов (количество неправильных вводов пароля), границах типов (даже если согласно спецификации можно в некоторое поле можно ввести ничем не ограниченное сверху целое число, это число так или иначе будет ограничено — максимальным значением целочисленного типа данных, который выбрал программист в реализации этой функции). Есть и другие границы, связанные с нефункциональными видами тестирования — производительности, конфигураций.
4. **Эмпирическое знание.** Некоторые значения могут выбираться чаще других или предполагать особое использование с точки зрения бизнес-логики приложения. Идеи таких тестов может подсказать человек, хорошо ориентирующийся в предметной области программы.

При анализе граничных значений и выделении классов эквивалентности для числовых параметров следует уделить особенное внимание результату вычислений. Такое разбиение и выбор значений для тестирования поможет найти важные ошибки. Какие ограничения накладываются на область значений, т.е. результат вычислений, и какие значения при этом должны принимать входные параметры? Какие нужно задать значения на вход, чтобы выйти за границы этой области? Например, если результат вычисления должен быть положительным, стоит выделить три класса эквивалентности и соответствующие им граничные значения:

- входные данные, при которых результат строго положительный (валидный класс);
- входные данные, при которых результат равен нулю (невалидный класс);
- входные данные, при которых результат отрицателен (невалидный класс).

Может оказаться, что значения, оказавшиеся при таком разбиении в невалидных классах, разрешены для ввода согласно спецификации. Это может быть ошибкой составления требований, на которую стоит указать бизнес-аналитикам.

Сочетания значений

Дефекты, зависящие от входных данных, можно поделить на те, которые возникают при конкретном значении одного параметра, и те, для возникновения которых нужно сочетание конкретных значений более чем одного параметра. Для обнаружения последних применяют комбинаторные техники тестирования, одной из которых является попарное тестирование (pairwise).

Комбинаторные техники можно применить, когда выделены классы эквивалентности для каждого параметра, и выбраны значения, на которых будут проводиться тесты для каждого параметра по отдельности. Для составления комбинаций можно воспользоваться несколькими стратегиями:

- «слабое» vs. «сильное» комбинирование — слабое позволит составить минимум тестов для обнаружения всех дефектов, возникающих на конкретном значении одного параметра, сильное предназначено для обнаружения дефектов, возникающих на «стыке» значений параметров;

- «нормальное» vs. «надежное» комбинирование — нормальное использует только валидные значения, выбранные для тестирования, надежное — все.

Это независимые взаимодополняющие характеристики, т.е., например, можно применить слабое нормальное комбинирование. Такая стратегия предполагает составление тестов, в которых хотя бы один раз встречаются все валидные значения каждого параметра, выбранного для тестирования.

Пример:

- параметр A может принимать валидные значения a1, a2, a3 и невалидные a_4, a_5;
- параметр B может принимать валидное значение b1 и невалидные b_2, b_3, b_4;
- параметр C может принимать валидные значения c1, c2, c3 и c4 и невалидное c_1.

Слабое нормальное комбинирование выдаст такие тесты:

- a1, b1, c1
- a2, b1, c2
- a3, b1, c3
- a1, b1, c4

Сильное нормальное комбинирование — все возможные комбинации валидных значений каждого из параметров:

- a1, b1, c1
- a1, b1, c2
- a1, b1, c3
- a1, b1, c4
- a2, b1, c1
- a2, b1, c2
- a2, b1, c3
- a2, b1, c4
- a3, b1, c1
- a3, b1, c2
- a3, b1, c3
- a3, b1, c4

Аналогично можно составить наборы тестов, используя стратегии слабого надежного и сильного надежного комбинирования (попробуйте сделать это для примера выше).

Очевидно, что при использовании сильного и/или надежного комбинирования количество тестов будет резко возрастать при увеличении количества значений какого-либо из параметров и, конечно, при увеличении количества самих параметров. **Техника попарного перебора** — один из способов уменьшить количество тестов, при этом попытавшись сохранить качество тестирования, т.е. свести к минимуму количество необнаруженных ошибок. Но применяя эту технику, важно понимать, что ошибки на стыке более чем двух значений параметров останутся ненайденными.

Другой способ уменьшить количество тестов — узнать, есть ли зависимости между входными параметрами, и учесть это в тестах. Это возможно далеко не всегда: часто тестирование черного ящика не позволяет «заглянуть внутрь». В этом случае можно попытаться выявить зависимости эмпирически и учесть их в комбинаторных тестах. Однако, есть риск ошибиться при выделении таких закономерностей.

Комбинаторные тесты можно и нужно составлять с помощью соответствующих инструментов, чтобы избежать человеческого фактора.