

ТЕСТОВЫЕ ТРЕБОВАНИЯ – ИСТОЧНИК ТЕСТОВЫХ ПРИМЕРОВ

Для тестирования программного обеспечения, кроме тестового окружения, необходимо определить проверочные задачи, которые будет выполнять система или ее часть. Такие проверочные задачи называют тестовыми примерами.

Каждый тестовый пример состоит из входных значений для системы, описания сценария работы примера и ожидаемых выходных значений.

ЦЕЛЬЮ выполнения любого тестового примера является либо продемонстрировать наличие в системе дефекта, либо доказать его отсутствие.

ТЕСТ-ТРЕБОВАНИЯ КАК ИСТОЧНИК ДЛЯ СОЗДАНИЯ ТЕСТОВЫХ ПРИМЕРОВ

Основным источником информации для создания тестовых примеров является различного рода документация на систему, например, функциональные требования и требования к интерфейсу.

Функциональные требования описывают поведение системы, как “черного ящика”, т.е. исключительно с позиций того, что должна делать система в различных ситуациях. Иными словами, функциональные требования определяют реакцию системы на различные входные воздействия. Рассмотрим это на примере функции вычисления неотрицательной степени n числа x [Котляров 2006].

СПЕЦИФИКАЦИЯ ПРОГРАММЫ

*На вход программа принимает два параметра: x - число, n – степень.
Результат вычисления выводится на консоль.*

Значения числа и степени должны быть целыми.

Значения числа, возводимого в степень, должны лежать в диапазоне – $[0, 999]$.

Значения степени должны лежать в диапазоне – $[1, 100]$.

Если числа, подаваемые на вход, лежат за пределами указанных диапазонов, то должно выдаваться сообщение об ошибке.

Начальный этап работы тестировщика заключается в формировании тест-требований, соответствующих функциональным требованиям. Основная цель тест-требований – определить, какая функциональность системы должна быть протестирована. В самом простом случае одному функциональному требованию соответствует одно тест-требование. Однако чаще всего тест-требования детализируют формулировки функциональных требований.

Тест-требования определяют, что должно быть протестировано, но не определяют, как это должно быть сделано. Например, для перечисленных выше функциональных требований можно сформулировать следующие тест-требования.

ТЕСТ-ТРЕБОВАНИЯ

Проверить, что если x находится в диапазоне $[0, 999]$ и n находится в диапазоне $[1, 100]$, то программа выдает правильный результат.

Проверить, что если x находится вне указанного диапазона или n находится вне указанного диапазона, то программа выдает сообщение об ошибке.

Проверить, что если x не является числом или n не является числом, то программа выдает сообщение об ошибке.

Особенности реализации тестового окружения и конкретные значения, подаваемые на вход системы, и ожидаемые на ее выходе определяются тестовыми примерами. Одному тест-требованию соответствует как минимум один тестовый пример.

РАЗРАБОТКА ТЕСТОВ

Определим области эквивалентности входных параметров.

Для x – числа, возводимого в степень, определим классы возможных значений:

1. $x < 0$ (ошибочное);
2. $x > 999$ (ошибочное);
3. x - не число (ошибочное);
4. $0 \leq x \leq 999$ (корректное);
5. Для n – степени числа:
6. $n < 1$ (ошибочное);
7. $n > 100$ (ошибочное);
8. n - не число (ошибочное);
9. $1 \leq n \leq 100$ (корректное).

АНАЛИЗ ТЕСТОВЫХ СЛУЧАЕВ

1. Входные значения: $(x = 2, n = 3)$ (классы 4, 8).

Ожидаемый результат: The power n of x is 8.

2. Входные значения: $\{(x = -1, n = 2), (x = 1000, n = 5)\}$ (классы 1, 2).

Ожидаемый результат: Error: x must be in $[0, 999]$.

3. Входные значения: $\{(x = 100, n = 0), (x = 100, n = 200)\}$ (классы 5, 6).

Ожидаемый результат: Error: n must be in $[1, 100]$.

4. Входные значения: $(x = \text{ADS}, n = \text{ASD})$ (классы 3, 7).

Ожидаемый результат: Error: Please enter a numeric argument.

5. Проверка на граничные значения:

5.1. Входные значения: ($x = 999$ $n = 1$).

Ожидаемый результат: The power n of x is 999.

5.2. Входные значения: $x = 0$ $n = 100$.

Ожидаемый результат: The power n of x is 0.

ВЫПОЛНЕНИЕ ТЕСТОВЫХ СЛУЧАЕВ

Запустим программу с заданными значениями аргументов.

ОЦЕНКА РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ПРОГРАММЫ НА ТЕСТАХ

В процессе тестирования производится оценка результатов выполнения путем сравнения получаемого результата с ожидаемым результатом.

ТИПЫ ТЕСТОВЫХ ПРИМЕРОВ

Для анализа возможных типов тестовых примеров рассмотрим функцию чтения имени файла. Спецификация данной функции следующая.

СПЕЦИФИКАЦИЯ ПРОГРАММЫ

Входным параметром для программы является имя файла.

Функциональные ограничения.

Имя файла не должно содержать ничего кроме символов латинского алфавита и цифр.

Имя файла не должно превышать 12 символов.

В случае неправильного задания имени файла программа должна выдавать сообщение об ошибке.

1. ДОПУСТИМЫЕ ДАННЫЕ

Чаще всего дефекты в программных системах проявляются при обработке нестандартных данных, не предусмотренных требованиями – при вводе неверных символов, пустых строк, слишком большой скорости ввода информации. Однако, перед поиском таких дефектов необходимо удостовериться в том, что программа корректно обрабатывает верные данные, предусмотренные спецификацией, т.е. проверить работу основных алгоритмов.

Тест-требование. Проверить программу на ввод допустимых данных.

Обычно для проверки допустимых данных достаточно одного тестового примера. Но функциональные требования могут определять различные группы допустимых данных, которые могут объединяться в классы

эквивалентности. В этом случае необходимо определять как минимум один тестовый пример для одного класса эквивалентности. Более подробно речь о классах эквивалентности пойдет далее.

2. ГРАНИЧНЫЕ ДАННЫЕ

Отдельный вид допустимых данных, передача которых в систему может вскрыть дефект – граничные данные, т.е. например, числа, значения которых являются предельными для их типа, строки предельной или нулевой длины и т.п. Обычно при помощи тестирования граничных условий выявляются проблемы с арифметическим сравнением чисел или с итераторами циклов.

Тест-требование. Проверить программу на ввод граничных данных.

Например, для тестирования функции чтения имени файла на граничных условиях можно определить два тестовых примера с минимальными и максимальными значениями полей в записи.

3. ОТСУТСТВИЕ ДАННЫХ

Дефекты могут проявиться и в случае, если системе не передается никаких данных или передаются данные нулевого размера. Например, для тестирования функции чтения имени файла при отсутствии данных можно вызвать ее, передав в качестве параметра – пустую строку.

Тест-требование. Проверить программу на отсутствие данных.

4. ПОВТОРНЫЙ ВВОД ДАННЫХ

В случае повторной передачи на вход системы тех же самых данных могут получаться различия в выходных данных, не предусмотренные в требованиях. Как правило, дефекты такого типа проявляются в результате того, что система не устанавливает внутренние переменные в исходное состояние или в результате ошибок округления.

Тест-требование. Проверить программу на повторный ввод данных.

5. НЕВЕРНЫЕ ДАННЫЕ

При проверке поведения системы необходимо не забывать проверять ее поведение при передаче ей данных, не предусмотренных требованиями – слишком длинных или слишком коротких строк, неверных символов, чисел за пределами вычислимого диапазона и т.п. Неверные данные, как и допустимые, также можно разделять на различные классы эквивалентности. Примером неверных данных для функции чтения имени файла может служить использование служебных символов.

Тест-требование. Проверить программу на ввод неверных данных.

6. РЕИНИЦИАЛИЗАЦИЯ СИСТЕМЫ

Механизмы повторной инициализации системы во время ее работы также могут содержать дефекты. В первую очередь эти дефекты могут проявляться в том, что не все внутренние данные системы после реинициализации придут в начальное состояние. В результате может произойти сбой в работе системы.

Тест-требование. Проверить работу программу при реинициализации системы.

7. УСТОЙЧИВОСТЬ СИСТЕМЫ

Под устойчивостью системы можно понимать ее способность выдерживать нештатную нагрузку, явно не предусмотренную требованиями. Например, сохранит ли система работоспособность после 10 тысяч вызовов.

Аналогичный анализ может быть сделан путем просмотра текста программы (если он доступен при тестировании) на основании отсутствия “истории” (хранимых данных) в реализации программы, т.е. данных, значение которых может меняться в зависимости от количества запусков программы. Таким образом, в ряде случаев тестирование может быть заменено анализом программного кода.

Тест-требование. Проверить программу на устойчивость.

8. НЕШТАТНЫЕ СОСТОЯНИЯ СРЕДЫ ВЫПОЛНЕНИЯ

Нештатные состояния среды выполнения (например, исчерпание памяти, дискового пространства или длительная нехватка процессорного времени) могут затруднять работу системы, либо делать ее невозможной. Основная задача системы в такой ситуации – корректно завершить или приостановить свою работу.

Тест-требование. Проверить программу при нештатном состоянии среды выполнения.

9. ГРАНИЧНЫЕ УСЛОВИЯ

В тестовых примерах, прямо соответствующих тест-требованиям обычно используются входные значения, находящиеся заведомо внутри допустимого диапазона. Один из способов проверки устойчивости системы на значениях, близких к предельным значениям – создавать для каждого входа как минимум три тестовых примера.

1. Значение внутри диапазона.
2. Минимальное значение.
3. Максимальное значение.

Для еще большей уверенности в работоспособности системы используют пять тестовых примеров.

1. Значение внутри диапазона.
2. Минимальное значение.
3. Минимальное значение + 1.
4. Максимальное значение.
5. Максимальное значение – 1.

Такой способ проверки называется проверкой на граничных значениях. Такая проверка позволяет выявлять проблемы, связанные с выходом за границы диапазона.

Например, если в функцию [Синицын 2006]:

```
char sum(char a, char b)
{
    return a+b;
}
```

вычисляющую сумму чисел а и b будут переданы значения 255 и 255, то в случае отсутствия специальной обработки ситуации переполнения сумма будет вычислена неверно.

Другая область, при тестировании которой полезно пользоваться проверкой на граничных значениях – индексы массивов.

Например, функция [Синицын 2006]:

```
void abs_array(char array[], char size)
{
    for (int i=1;i<=size;i++)
    {
        array[i] = abs(array[i]);
    }
    return;
}
```

заменяющая значение каждого элемента переданного ей массива на значение по модулю содержит ошибку в цикле for, которая может быть легко обнаружена при передаче в функцию массива единичного размера.

10. ПРОВЕРКА РОБАСТНОСТИ (ВЫХОДА ЗА ГРАНИЦЫ ДИАПАЗОНА)

РОБАСТНОСТЬ СИСТЕМЫ – это степень ее чувствительности к факторам, не учтенным на этапах ее проектирования, например, к неточности основного алгоритма, приводящего к ошибкам округления при вычислениях, сбоям во внешней среде или к данным, значения которых находятся вне допустимого диапазона. Чаще всего под робастностью программных систем понимают именно устойчивость к некорректным данным. Система должна быть способна корректно обрабатывать такие данные путем выдачи соответствующих сообщений об ошибках, сбое и отказы системы на таких данных недопустимы.

Для тестирования робастности к тестовым примерам, рассмотренным в предыдущем разделе, добавляются еще два тестовых примера.

1. Минимальное значение – 1.
2. Максимальное значение + 1.

проверяющие поведение системы за границей допустимого диапазона, а также в случае тестирования операций сравнения, дополнительно дающие гарантию того, что в них не допущена опечатка.

Таким образом, если изобразить допустимый интервал, как на Рис.13 [Синицын 2006], то можно видеть, что для тестирования интервальных значений достаточно 7 тестовых примеров – пяти допустимых и двух на робастность.

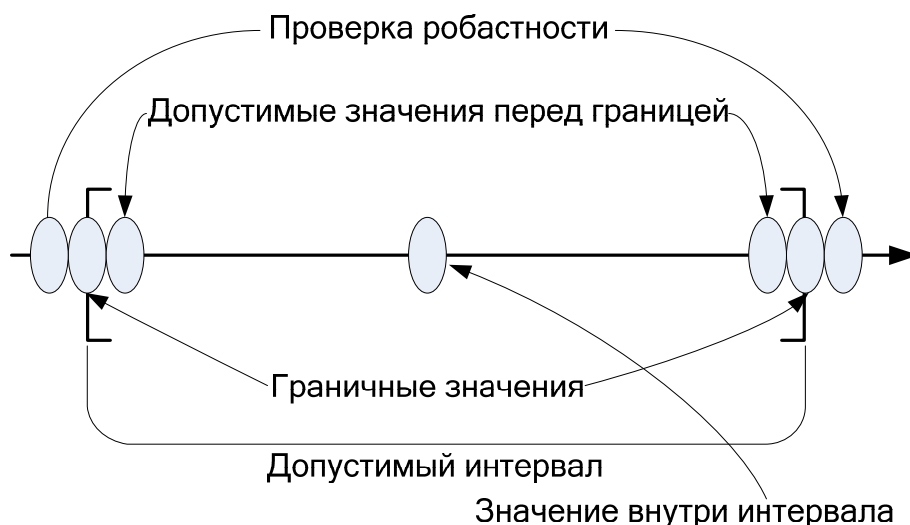


Рис.13 Рекомендуемые проверочные значения

В литературе часто встречается утверждение, что значение внутри интервала является избыточным и его тестирование не требуется. Однако, проверка внутреннего значения является полезной как минимум с психологической точки зрения, а также в случае если интервал ограничен сложными граничными условиями.

Также рекомендуется отдельно проверять значение 0 (даже если оно находится внутри интервала), т.к. зачастую это значение обрабатывается некорректно (например, в случае деления на 0).

КЛАССЫ ЭКВИВАЛЕНТНОСТИ

При разработке тестовых примеров может возникнуть такая ситуация, в которой различные входные значения приводят к одним и тем же реакциям системы. Если при этом такие входные значения имеют что-то общее, то возможно объединение таких значений в классы эквивалентности, т.е. выполнение эквивалентного разбиения множества допустимых входных значений.

Разбиение на классы эквивалентности – в первую очередь, способ уменьшения необходимого числа тестовых примеров. Обычно, если в тест-требованиях специально не оговорено иное, при тестировании достаточно выполнить только один тестовый пример для каждого класса эквивалентности. Разбиение на классы эквивалентности особенно полезно, когда на вход системы может быть подано большое количество различных значений; тестирование каждого возможного значения привело бы к слишком большому объему тестирования.

Рассмотренные выше граничные условия могут служить примером классов эквивалентности.

1. Значение из середины интервала.
2. Граничные значения.
3. Недопустимые значения за границами интервала.

Таким образом, тестирование граничных условий и робастности является частным случаем тестирования с использованием классов эквивалентности – вместо того, чтобы тестировать все недопустимые значения, выбираются только соседние с граничными значениями.

При определении классов эквивалентности следует руководствоваться следующими правилами.

- ✓ Всегда будет, по меньшей мере, два класса: корректный и некорректный.
- ✓ Если входное условие определяет диапазон значений, то, как правило, бывает три класса: меньше чем диапазон, внутри диапазона и больше чем диапазон. (Значения на концах диапазона могут трактоваться как граничные значения.)
- ✓ Если элементы диапазона обрабатываются по-разному, то каждому варианту обработки будут соответствовать разные требования.

Другим примером разбиения на классы эквивалентности может служить тестирование открытия файла по его имени [Синицын 2006]. В результате тестирования необходимо определить, все ли варианты имен обрабатываются системой согласно следующим тест-требованиям.

1. Проверить, что в случае присутствия в имени файла символов, не являющимися буквами латинского алфавита и цифрами, система выводит сообщение об ошибке.
2. Проверить, что в том случае, когда длина имени файла превышает 11 символов, система выдает сообщение об ошибке.
3. Проверить, что система не различает регистр символов имени при открытии файла.
4. Проверить, что при открытии файлов с именами, не противоречащими требованиям 1-3, система открывает файл.

Входными значениями тестового примера являются различные имена файлов, выходными – реакция системы (ошибка или успешное открытие).

Можно выделить следующие классы эквивалентности.

ПО ДЛИНЕ ИМЕНИ:

Длина имени меньше 11 символов.

Длина имени равна 11 символам.

Длина имени больше 11 символов.

ПО СИМВОЛАМ:

Имя, состоящее из цифр и букв смешанного регистра.

Имя, состоящее из цифр и букв нижнего регистра.

Имя, состоящее из цифр и букв верхнего регистра.

Имя, состоящее только из цифр.

Имя, состоящее только из букв.

Имя, включающее знаки препинания (не буквенно-цифровые символы).

Имя, включающее управляющие символы (не буквенно-цифровые символы).

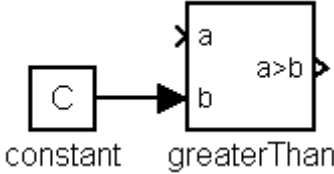
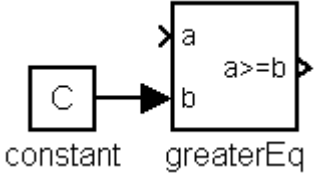
Эти классы эквивалентности иллюстрируют, что проверки на границах интервалов применимы не только для тестирования арифметических операций и операций сравнения. Практически для любых данных, даже текстовых, можно определить “минимальные” и “максимальные” допустимые значения.

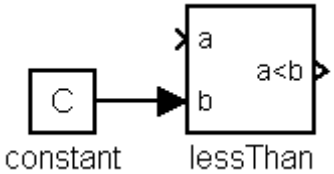
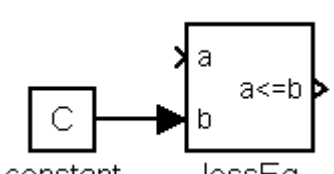
ТЕСТИРОВАНИЕ ОПЕРАЦИЙ СРАВНЕНИЯ ЧИСЕЛ

Разбиение на классы эквивалентности широко используется при тестировании корректности реализации арифметических операций и операций сравнения. Каждую операцию можно рассматривать как блок с входами – значениям и выходом – результатом операции. Для ее тестирования выполняется разбиение диапазона изменения переменных на входах блока на классы эквивалентности и методом анализа граничных значений этих переменных.

В таблице 3 [Синицын 2006] приведены тестовые наборы для блоков реализующих операции сравнения, в случае, когда на один из входов блока подаётся константа.

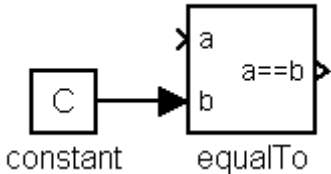
Таблица 3. Блоки сравнения и определённые для них тестовые наборы

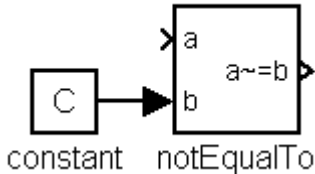
greaterThan блок. Реализует операцию сравнения a>b (b – константа, на входе a может быть переменная числового типа)						greaterEq блок. Реализует операцию сравнения a>=b (b – константа, на входе a может быть переменная числового типа)					
											
№ набора	1	2	3*	4	5	№ набора	1	2	3*	4	5
Вход a	b - d	b + d	b	min	max	Вход a	b - d	b + d	b	min	max
Выход	F	T	F	F	T	Выход	F	T	T	F	T

lessThan блок. Реализует операцию сравнения a<b (b – константа, на входе a может быть переменная числового типа)						lessEq блок. Реализует операцию сравнения a<=b (b – константа, на входе a может быть переменная числового типа)					
											
№ набора	1	2	3*	4	5	№ набора	1	2	3*	4	5

Вход a	b - d	b + d	b	min	max
Выход	T	F	F	T	F

Вход a	b - d	b + d	b	min	max
Выход	T	F	T	T	F

<p>equalTo блок. Реализует операцию сравнения a=b (b – константа, на входе a может быть переменная числового типа)</p> 				
№ набора	1	2	3	4
Вход a	≠ b	b	min	max
Выход	F	T	F	F

<p>notEqualTo блок. Реализует операцию сравнения a≠b (b – константа, на входе a может быть переменная числового типа)</p> 				
№ набора	1	2	3	4
Вход a	≠ b	b	min	max
Выход	T	F	T	T

* тестовый набор реализуем только, если переменная на входе a – переменная целого типа.

В приведённых тестовых наборах используются следующие обозначения:

- ✓ d – шаг изменения переменной на входе a. Если переменная на входе a – переменная целого типа, то d равно 1;
- ✓ min – минимальное значение переменной на входе a;
- ✓ max – максимальное значение переменной на входе a.

ЛИТЕРАТУРА

[Синицын 2006] – Синицын С.В., Налютин Н.Ю. Верификация программного обеспечения. Курс лекций. Московский инженерно-физический институт. М. 2006.

[Котляров 2006] – В.П. Котляров, Т.В. Коликова. Основы тестирования программного обеспечения. Учебное пособие. М.: Интернет-Университет Информационных технологий.