Тестирование

"Тестирование программ может использоваться для демонстрации наличия ошибок,
но оно никогда не покажет их отсутствие"
Эдсгер Вибе Дейкстра

Необходимо различать понятия тестирование и отладка.

Тестирование – это процесс исполнения программы с целью обнаружения ошибок (Г. Майерс).

Тестирование направлено на выявление ошибок в программе, которая считается работающей. В то время как, *отладкой* называют процесс локализации и исправления обнаруженных ошибок.

Программа подвергается постоянному тестированию на протяжении всего процесса работы над ней: начиная от стадии разработки и заканчивая внедрением ее в эксплуатацию.

Разработчик должен сопровождать тестированием каждый небольшой блок кода. Только в процессе непосредственной работы над программой разработчик лучше всего понимает функциональное назначение каждого блока. Систематическое тестирование отдельных составляющих блоков, объем которых постепенно увеличивается, позволяет создавать в результате более эффективные программы. Тестирование всей программы после того как написан весь код отнимает много времени и усилий, при этом внесенные изменения являются, как правило, менее продуманными.

Основным условием для тестирования является наличие известных наборов исходных данных и соответствующих им результатов выходных данных. Укажем некоторые рекомендации к тестированию программ [2].

1) Все результаты тестирования необходимо строго фиксировать на бумаге, так чтобы их можно было воспроизвести после того как будут внесены изменения в программу.

2) Необходимо тестировать граничные условия. Например, необходимо постоянно проверять, что тело цикла повторяется нужное число раз, а условное выражение корректно разветвляет вычисления.

3) Необходимо тестировать пред- и постусловия. Идея состоит в том, чтобы проверять условия до (предусловие) или после (постусловие) выполнения некоторого блока кода. Другими словами, прежде чем использовать данные в вычислениях, необходимо удостовериться в их корректности. Например, при вычислении частного двух чисел делитель не должен быть равен 0; индекс массива не должен превышать допустимый диапазон.

4) Следует проверять коды возвратов функций. Например, если по каким-либо причинам вызов библиотечной функции fopen(), осуществляющей открытие файла, завершилось неудачей, то дальнейшая работа с данным файлом должна быть прервана.

Фатальные ошибки программного обеспечения

Чтобы проиллюстрировать важность этапа тестирования, перечислим только некоторые ошибки в программном обеспечении, которые привели к негативным последствиям.

В 2012 году сбой компьютерной системы чуть не привел инвестиционную компанию "Knight Capital" к банкротству. Фирма потеряла полмиллиарда долларов через полчаса после того, как ошибка программного обеспечения позволила компьютерам покупать и продавать миллионы акции без надзора человека.

В ноябре 2000 года в Панаме 20 человек получили переоблучение и восемь человек погибли из-за того, что программа для планирования радиационной терапии неправильно рассчитывала дозы облучения для пациентов.

В ноябре 1998 года на борту американского крейсера Yorktown произошел инцидент. Член команды по ошибке вместо значимого числа ввел 0 ("нуль"), что привело к ошибке деления на нуль, ошибка разрослась и, в конце концов, силовая установка оказалась выведена из строя.

4 июня 1996 года в воздухе взорвалась новая ракета-носитель "Ariane 5". Ракета самоликвидировалась через 40 секунд после старта из-за цепочки компьютерных ошибок. Началом этой цепочки послужило переполнение буфера, связанное с преобразованием типов. Навигационный пакет для "Ariane 5" был унаследован от "Ariane 4" и не прошел тщательного тестирования. Новая ракета имела большую скорость, и, соответственно, навигационным программам приходилось иметь дело с большими числами.

Стратегии тестирования

Существуют различные способы классификации методов и подходов к тестированию. В данном разделе мы приведем две стратегии тестирования программного обеспечения, которые являются наиболее распространенными.

При тестировании с использованием *стратегии "белого ящика"* (иногда можно встретить термин "прозрачный ящик") полагают, что структура программного обеспечения известна, этим и объясняется название стратегии. Это тестирование выполняют непосредственные разработчики программного обеспечения или программисты. Возможно, что к тестированию привлекаются также отдельные специалисты, занимающиеся тестированием программ (тестировщики), которые имеют доступ к исходному коду программы. В данном случае тесты подбирают таким образом, чтобы пройти хотя бы один раз по каждой ветви алгоритма.

При тестировании с использованием *стратегии "черного ящика"* полагают, что структура программного обеспечения неизвестна, то есть программа рассматривается как "черный ящик" при этом известны только наборы входных и выходных данных. В данном случае, тестовые данные формируют только на основе спецификации программы, без учета информации о ее структуре. Это тестирование выполняют тестировщики на этапе испытаний и внедрения в эксплуатацию, и, как правило, оно направлено на проверку граничных условий, проверку работоспособности программы на больших объемах данных, и на некорректных входных данных, также важным является проверка программы в нормальных условиях.

При тестировании сложных программных систем стратегия "белого ящика" не позволяет выявить пропущенный маршрут, поэтому, возможно, что некоторые ошибки в программе останутся не обнаруженными. Также эта стратегия не позволяет выявить ошибки, связанные с обрабатываемыми входными данными. В свою очередь, при использовании стратегии "черного ящика" невозможно подобрать такие тестовые наборы, чтобы выполнить проверку всех возможных комбинаций исходных данных. В этой связи, на практике, как правило, используют комбинации двух стратегий.

В заключение отметим, что даже после внедрения в эксплуатацию процесс тестирования не прекращается. Конечным пользователям удается обнаруживать новые ошибки в процессе всестороннего использования программного продукта для решения конкретных задач.

В качестве примера составим тест для программы, которую чаще всего рассматривают в литературе, посвященной тестированию.

Пример 1

Вводятся значения трех сторон треугольника. Необходимо определить вид треугольника и вычислить его площадь.

На рисунке 1 приводим псевдокод алгоритма решения поставленной задачи. На первый взгляд программа может показаться очень простой. Однако обращаем внимание

на то, что даже в такой простой программе стоит особое внимание уделять корректности введенных данных. Вид треугольника определим на основе сравнения сторон треугольника. Площадь треугольника вычислим по формуле Герона. В

таблице 1 приведен пример тестов для данной программы. К какой стратегии относится представленный тест?

```
Ввести три стороны треугольника a, b, c.
Если a или b или c не являются числом
        Вывести сообщение «Некорректный ввод. Необходимо ввести число.»
Иначе
        Если  a<0 или b<0 или c<0
              Вывести сообщение «Некорректный ввод. Числа должны быть положительнь
        Иначе
                Если a>b+c  или b>a+c или c>b+c
                        Вывести сообщение «Не является треугольником.»
                Иначе
                     Если a == b и b == c
                        Вывести сообщение «Треугольник является равносторонним.»
                     Иначе
                        Если a == b или b == c или a == c
                            Вывести сообщение «Треугольник является равнобедренным.»
                        Иначе
                            Вывести сообщение «Треугольник является разносторонним.»
                        Все-если
                        Все-если
                        Вычислить полупериметр треугольника P=(a+b+c)/2
                        Вычислить площадь треугольника S=sqrt(P(P-a)(P-b)(P-c))
                       Вывести S.
                Все-если
          Все-если
Все-если
Конец алгоритма
```

Рис. 1.

№	Цель тестирования	Значения			Ожидаемый результат	Результат работы программы
		a	b	c		
1	Корректность введенных данных	d	1	2	Некорректный ввод.	
2	Корректность введенных данных	0	0	0	Некорректный ввод.	
3	Корректность введенных данных	1	-3	2	Некорректный ввод.	
4	Корректность введенных данных	5	1	3	Не является треугольником.	
5	Корректность введенных данных	8	2	6	Не является треугольником.	
6	Результат	8	10	6	Разносторонний треугольник, S = 24	
7	[English] Результат	5	10	6	Разносторонний треугольник, S = 11,399	
8	Результат	7	7	7	Равносторонний треугольник, S = 21,218	
9	Результат	4	3	4	Равнобедренный треугольник, S = 5,562	
10	Результат	3	2	2	Равнобедренный треугольник, S = 1,984	

Рис. 2.