что его реализация не отнимает дополнительного времени у разработчика, так как исполнение команд в объектах – это простое перенаправление параметров на уже описанные в объекте методы. А программирование консоли требует незначительных усилий.

В дальнейшем, лишь незначительно изменив условия поступления команд, можно обрабатывать командные файлы. Таким образом, возможна автоматическая инициализация системы, что значительно проще программной, так как позволяет изменять параметры после компиляции программы. Кроме того, возможно создание тестовых последовательностей, отработка которых каждый раз при старте системы гарантирует ее работоспособность. Реализация этих дополнений значительно упростит работу разработчиков программных модулей многократного использования.

Реализовать консольное управление можно не только в графических системах визуализации. В большинстве систем, связанных с обработкой сложных моделей, хранящихся во внешних ресурсах может возникнуть необходимость приостановить работу системы, изменить параметры системы или модели и запустить ее опять. По мере накопления информации об ошибках и неточностях, а также точных методах их исправления производится перекомпиляция. Такой подход сокращает временные издержки на отладку любой системы.

## ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Карасик Е. А.

Белорусский государственный университет информатики и радиоэлектроники

г. Минск, ул. П. Бровки, 6

**Аннотация:** обзорная статья по основным методам и стратегиям тестирования программного обеспечения.

**Ключевые слова:** тестирование; программное обеспечение;

Одними из основных методов повышения надежности ПО, являются тестирование, отладка и верификация, которые состоят в проверке КП на соответствие заданной спецификации [1].

Тестирование программного обеспечения охватывает целый ряд видов деятельности. Сюда входят постановка задачи для теста, проектирование, написание тестов, тестирование тестов и, наконец, выполнение тестов, и изучение результатов тестирования. Решающую роль играет проектирование тестов. Существует целый спектр подходов к проектированию тестов [2].

При тестировании ПО, используя подход функционального тестирования, программу рассматривают как «черный ящик» (ее исходный текст не используется). Происходит проверка соответствия поведения программы ее спецификации. Критерием полноты тестирования при таком подходе является полный перебор всех возможных значений входных данных, (исчерпывающее тестирование), что является невыполнимой задачей.

При структурном тестировании программа рассматривается как "белый ящик" (т.е. ее текст открыт для пользования). Происходит проверка логики программы. Полным тестированием в этом случае будет такое, которое приведет к перебору всех возможных путей на графе передач управления программы (ее управляющем графе). Даже для средних по сложности программ числом таких путей может достигать десятков тысяч. Поэтому при структурном тестировании необходимо использовать другие критерии его полноты, позволяющие достаточно просто контролировать их выполнение, но не дающие гарантии полной проверки логики программы.

Но даже если предположить, что удалось достичь полного структурного тестирования некоторой программы, в ней, тем не менее, могут содержаться ошибки, т.к.

1. программа может не соответствовать своей внешней спецификации, что в частности, может привести к тому, что в ее управляющем графе окажутся пропущенными некоторые необходимые пути;
2. не будут обнаружены ошибки, появление которых зависит от обрабатываемых данных (т.е. на одних исходных данных программа работает правильно, а на других – с ошибкой).

Таким образом, ни структурное, ни функциональное тестирование не может быть исчерпывающим. Поэтому при построении тестов необходимо различные комбинации этих двух подходов.

Также следует отметить метод символического тестирования [3]. Этот метод позволяет проводить тестирование без конкретных исходных данных, т.е. использовать такой язык программирования, где вместо реальных данных фигурируют произвольные символы. Такое символическое выполнение является естественным расширением нормального (числового) выполнения. В этом случае в качестве результатов выдаются формулы с символическими значениями (т.е. арифметические формулы заменяются алгебраическими).

Существуют два вида символического выполнения программ:

1. по заданному пути программы необходимо определить, каким условиям должны удовлетворять входные значения, чтобы при выполнении программы на этих данных управление передавалось бы именно по этому пути. Символическое выполнение по заданному пути программы можно осуществлять как с начала, так и с конца.
2. по зафиксированным ограничениям на символические входные данные определить путь (или пути) программы, который будет выполнен на этих данных, и символические значения переменных в конце пути.

Применение символического выполнения имеет следующие основные достоинства:

а) символическое выполнение первого вида, которое применяется для генерации тестов, позволяет образовать тесты, учитывая логическую структуру программы. В этом случае удается всесторонне проверить программу, например, автоматически составить тесты так, чтобы на них были выполнены все ветви программы;

б) символическое выполнение второго вида дает возможность одним выполнением программы на символических входных данных заменить весьма большое число прогонов на конкретных значениях. Если при этом удается разбить область входных значений программы на классы, где каждый класс содержит конкретные значения, эквивалентные с точки зрения логики программы, то выполнение программы на символических значениях, представляющих эти классы, довольно часто оказывается полным (и конечным) перебором всех возможных входных значений.

Вторым по важности аспектом тестирования (после проектирования тестов) является последовательность слияния всех модулей в систему или программу [2,3].

Основными стратегиями тестирования являются восходящее и нисходящее тестирование. Но применение этих двух стратегий в "чистом" виде обычно не применяется. Существует большой выбор различных подходов, которые могут быть использованы для слияния модулей в более крупные единицы, более подробно это описано в работе [2]

Из-за ограниченности, ресурсы тестирования используются, прежде всего, для обнаружения наиболее опасных ошибок в наиболее важных режимах функционирования программ. С этой целью последовательно применяются методы

тестирования: статический, детерминированный, стохастический и в реальном масштабе времени.

Статическое тестирование является наиболее формализованным и автоматизируемым методом проверки корректности программ. В качестве эталона применяются правила структурного построения программных модулей и обработки данных, конкретизированные для проекта в целом. Проверка степени выполнения этих правил проводится без исполнения объектного кода программы путем формального анализа текста программы на языке программирования.

При детерминированном тестировании контролируется каждая комбинация исходных эталонных данных и соответствующая ей комбинация результатов функционирования программы. Это позволяет выявлять отклонение результатов от эталона с конкретным фиксированием все значений исходных и результирующих данных, при которых это отклонение обнаружено.

В сложных программах невозможно перебрать все комбинации исходных данных и проконтролировать результаты функционирования программы на каждой из них. В таких случаях применяется стохастическое тестирование, при котором исходные данные задаются множествами случайных величин с соответствующими распределениями и для сравнения полученных результатов используются распределения случайных величин.

В процессе тестирования в реальном масштабе времени проверяется исполнение программ, и обработка исходных данных с учетом времени их поступления, длительности и приоритетности обработки, динамики использования памяти и взаимодействия с другими программами и т.д.

Литература

1. Липаев В.В. Тестирование программ. – М.: Радио и связь, 1986
2. Майерс Г. Надежность программного обеспечения. – М.: Мир, 1980
3. Бахтизин В. В., Иьуду К. А., Савкин В. В. Методы тестирования и верификации программ. – М.: Машиностроение, 1984

## СИСТЕМА КОНТРОЛЯ ЗНАНИЙ

Тиунчик Д.В., Лобов С.Д.
Брестский государственный университет им. А.С.Пушкина
Брест, Беларусь

**Аннотация:** в работе рассмотрено построение пакета контролирующих программ в среде Windows 9x. Структура и особенности пакета, его возможно-