**Тестирование разрабатываемой системы**

При создании программного проекта около 50% общего времени и более 50% общей стоимости расходуется на проверку (тестирование) разрабатываемой программы или системы. Кроме того, доля стоимости тестирования в общей стоимости программ имеет тенденцию возрастать при увеличении сложности комплексов программ и повышения требований к их качеству.

Учитывая это, при отработке технологии тестирования программ следует четко выделять определенное (по возможности не очень большое) число правил отладки, обеспечивающих высокое качество программного продукта и снижающих затраты на его создание.

Тестирование – очень важный и трудоемкий этап процесса разработки программного обеспечения, так как правильное тестирование позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ. Процесс разработки программного обеспечения предполагает три стадии тестирования: автономное, комплексное и системное, каждая из которых соответствует завершению соответствующей части системы. Различают два подхода к формированию тестов: структурный и функциональный. Каждый из указанных подходов имеет свои особенности и области применения.

Недостаточно выполнить проектирование и кодирование программного обеспечения, необходимо также обеспечить его соответствие требованиям и спецификациям. Многократно проводимые исследования показали, что чем раньше обнаруживаются те или иные несоответствия или ошибки, тем больше вероятность их правильного исправления и ниже его стоимость.

Тестирование – это процесс исполнения программы с целью обнаружения ошибок. Никакое тестирование не может доказать отсутствие ошибок в хоть сколько-нибудь сложном программном обеспечении. Для такого программного обеспечения выполнение полного тестирования, т. е. задания всех возможных комбинаций исходных данных, становится невозможным, а, следовательно, всегда имеется вероятность того, что в программном обеспечении остались не выявленные ошибки. Однако соблюдение основных правил тестирования и научно обоснованный подбор тестов может уменьшить их количество.

Процесс разработки программного обеспечения, в том виде, как он определяется в современной модели жизненного цикла программного обеспечения, предполагает три стадии тестирования:

* автономное тестирование компонентов программного обеспечения;
* комплексное тестирование разрабатываемого программного обеспечения;
* системное или оценочное тестирование на соответствие основным критериям качества.

В качестве методов автономного тестирования компонентов системы используются различные подходы:

* Метод ручного контроля;
* Тестирование по принципу «белого ящика»;
* Тестирование по принципу «черного ящика».

Согласно [Иванова], существуют следующие подходы для комплексного тестирования компонентов ПО:

* Восходящее тестирование;
* Нисходящее тестирование.

Восходящее тестирование предполагает, что каждый компонент тестируют отдельно на соответствие имеющимся спецификациям на него, затем собирают оттестированные компонент в компоненты более высокой степени интеграции и тестируют их. При этом проверяют интерфейсы между компонентами, используемые для подключения компонентов более низкого уровня иерархии. И так далее, пока не будет собран весь программный продукт.

Нисходящее тестирование заключается в том, что, когда проектирование какого-либо компонента заканчивается, его кодируют и передают на тестирование. При его тестировании все вызываемые им компоненты заменяют компонентами, которые в той или иной степени имитируют поведение вызываемых компонентов. Такие компоненты принято называть «заглушками». В отличие от тестирующих программ заглушки очень просты, например, они могут просто фиксировать, что им передано управление. Часто заглушки просто возвращают какие-либо фиксированные данные. Как только тестирование основного компонента завершено, к нему подключают компоненты, непосредственно им вызываемые, и необходимые заглушки, а затем проводят их совместное тестирование. Далее последовательно подключают следующие компоненты, пока не будет собрана вся система.

При разработке репозитория инсталляционных пакетов первым делом разрабатывались классы моделей сущностей, которые были выделены в результате объектной декомпозиции, затем производилась разработка контроллеров, которые описывают бизнес-процессы системы. Параллельно с разработкой моделей и контроллеров велась разработка пользовательского интерфейса. При этом, контроллеры, которые должны были вызываться элементами пользовательского интерфейса были заменены на «заглушки». Таким образом, тестирование репозитория инсталляционных пакетов происходило с использованием как восходящего подхода, так и нисходящего подхода.

**Тестирование методом ручного контроля**

**Тестирование методом инспекции исходного текста**

Данный метод тестирования представляет собой один из методов ручного контроля. Согласно [Иванова], инспекция исходного текста представляет собой набор процедур и приемов обнаружения ошибок при изучении текста группой специалистов. В эту группу входят: автор программы, проектировщик, специалист по тестированию и координатор - компетентный программист, но не автор программы. Общая процедура инспекции предполагает следующие операции:

* участникам группы заранее выдается листинг программы и спецификация на нее;
* программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;
* программа анализируется по списку вопросов для выявления исторически сложившихся общих ошибок программирования.

Рассмотрим тестирование данным способом на примере проверки файлов на дублирование. Изначально, для определения дублирования файла, от нового файла бралась хэш-сумма, и производился поиск в базе данных файла с такой же хеш-суммой. В ходе ответов на вопросы инспектора было обнаружено, что одной хеш-суммы для проверки на дублирование недостаточно, так как для разных файлов возможны коллизии. Для исправления данной ошибки было принято решение осуществлять проверку на дублирование по двум параметрам: по хэш-сумме и по размеру файла.

**Тестирование методом сквозных просмотров**

Данный метод, согласно [Иванова], представляет собой набор способов обнаружения ошибок, осуществляемых группой лиц, просматривающих текст программы, и состоит из следующих этапов:

1. Участникам группы заранее выдается листинг программы, блок схема и спецификация на нее;
2. Участникам заседания предлагается несколько тестов, написанных на бумаге, и тестовые данные подвергаются обработке в соответствии с логикой программы (каждый тест мысленно выполняется);
3. Программисту задаются вопросы о логике проектирования и принятых допущениях;
4. Состояние программы (значения переменных) отслеживается на бумаге или доске.

Рассмотрим тестирование данным методом на примере процедуры добавления нового файла в каталог программного обеспечения. Алгоритм данной процедуры представлен на рисунке 33, листинг приведен в приложении 111. Спецификация алгоритма:

Пользователь отправляет на сервер несколько файлов. Процедура должна проверить наличие прав у пользователя на загрузку файлов. Если у пользователя нет прав на загрузку файлов, то необходимо вывести сообщение об ошибке. Если у пользователя есть права – то для каждого файла необходимо убедится, что такого файла в системе еще нет. Если файл уже существует, то необходимо вывести об этом сообщение пользователю. Если такого файла нет, то файл нужно сохранить на диск и добавить запись о файле в базу данных. Затем, необходимо прочитать свойства файла и, если они заполнены, сохранить их в базу данных. В конце, необходимо вывести пользователю информацию о файлах, которые были добавлены в систему и предложить пользователю заполнить информацию о файлах, свойства который получить не удалось.

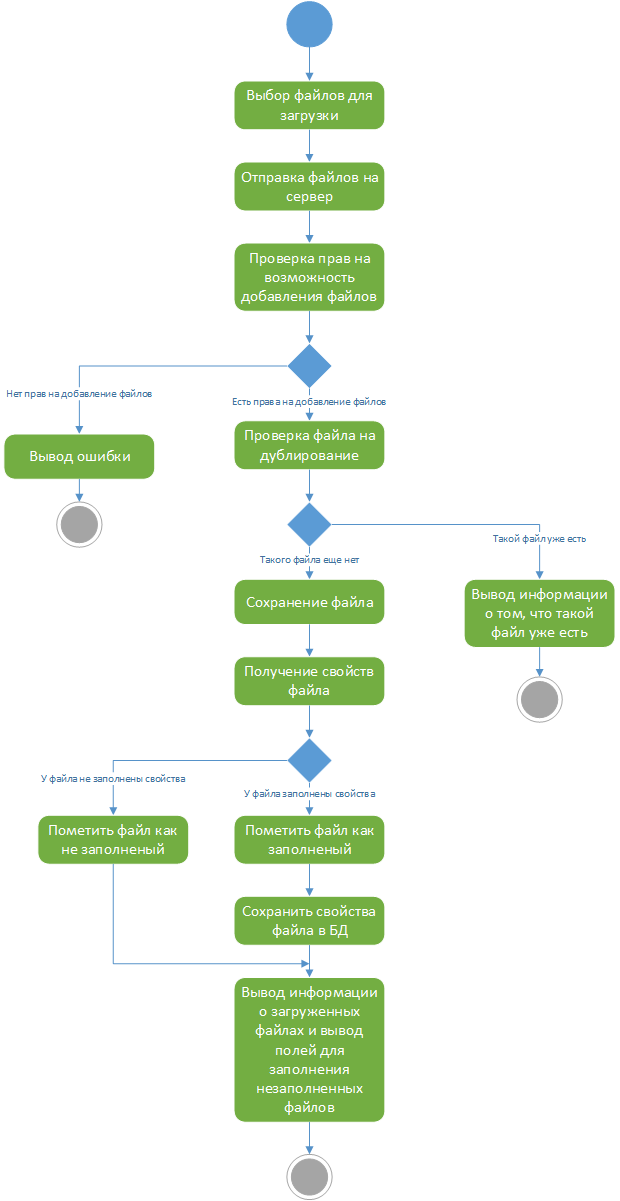


Рисунок 33 – Схема процедуры добавления файла в каталог программного обеспечения

В результате анализа схемы и листинга программы была обнаружена ошибка. При отправке пользователем файлов для добавления в каталог программного обеспечения они помещаются во временную папку. Затем, каждый файл открывается для копирования содержимого файла в новый файл рабочей директории приложения. Ошибка заключалась в том, что после завершения копирования содержимого, файл не закрывался, что приводило к невозможности удаления временных файлов.

**Тестирование по принципу «белого ящика»**

Согласно [Иванова], Стратегия тестирования по принципу «белого ящика», или стратегия тестирования, управляемая логикой программы (с учетом алгоритма), позволяет проверить внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы.

Стратегия «белого ящика» включает в себя следующие методы тестирования:

* Покрытие операторов
* Покрытие решений
* Покрытие условий
* Покрытие решений/условий
* Комбинаторное покрытие условий

При тестировании по принципу «белого ящика» использовался метод комбинаторного покрытия условий, так как данный метод охватывает решений и условий. Данный метод требует создания такого множества тестов, чтобы все возможные комбинации результатов условий в каждом решении и все операторы выполнялись, по крайней мере, один раз.

Рассмотрим тестирование методом комбинаторного покрытия условий на примере процедуры добавления нового файла в каталог программного обеспечения, алгоритм которого представлен на рисунке 33. В данном возможны следующие комбинации:

1. У пользователя нет прав на добавление файлов;
2. У пользователя есть права на добавление файлов;
3. Файл уже существует;
4. Файл еще не существует;
5. У файла заполнены свойства;
6. У файла не заполнены свойства.

Для каждой комбинации необходимо провести тестирование, то есть необходимо провести 6 тестов.

В результате проведенных тестов, было обнаружено, что в случае загрузки файлов форматов, отличных от Portable Executable (ОС Windows), возникала ошибка, приводящая к неожиданному завершение работы. Для исправления данной ошибки в функцию получения свойств файла была добавлена проверка файла на принадлежность формату Portable Executable. Алгоритмы работы функции получения свойств файла до и после исправления ошибки представлены на рисунках 33 и 34 соответственно.

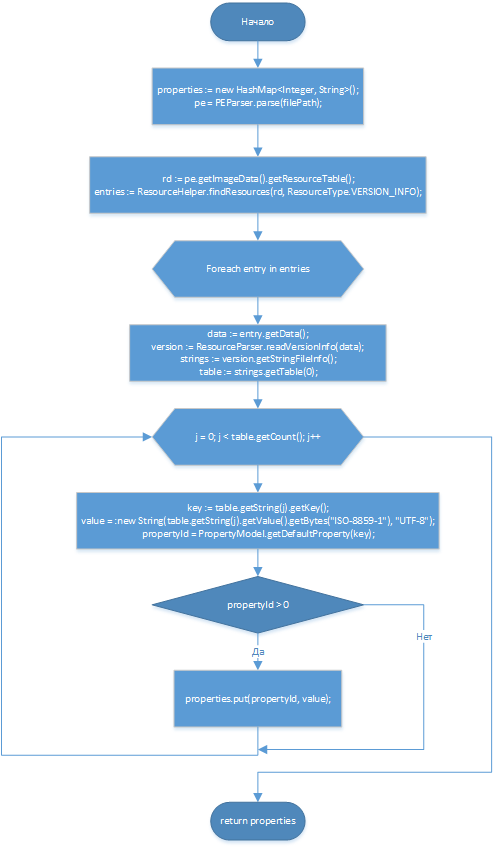


Рисунок 33 – Блок схема алгоритма получения свойств файла до исправления ошибки

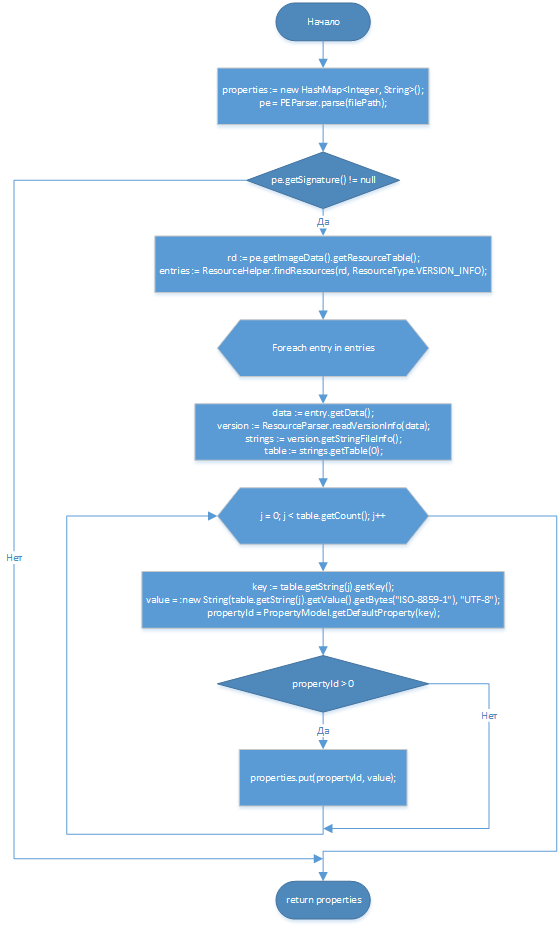


Рисунок 34 – Блок схема алгоритма получения свойств файла после исправления ошибки

**Тестирование по принципу «черного ящика»**

При тестировании данным методом программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Для обнаружения всех ошибок в программе, необходимо выполнить исчерпывающее тестирование, т. е. тестирование на всех возможных наборах данных. Для программ, в которых исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности. Очевидно, что проведение исчерпывающего тестирования для подавляющего большинства случаев невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование, которое ограничивается прогонами программы на небольшом подмножестве всех возможных входных данных. Этот вариант не дает гарантии отсутствия отклонений от спецификаций.

Согласно [Иванова], различают следующие методы формирования тестовых наборов:

* эквивалентное разбиение;
* анализ граничных значений;
* анализ причинно-следственных связей;
* предположение об ошибке.

Метод эквивалентного разбиения заключается в следующем. Исходные данные разбиваются на конечное число групп – классов эквивалентности. Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок: если набор какого-либо класса обнаруживает некоторую ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот. Так же, каждый тест должен включать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов.

Анализ граничных значений. Граничные условия - это ситуации, возникающие на, выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения следующим:

* выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется таким образом, чтобы проверить тестом каждую границу этого класса;
* при разработке тестов рассматриваются не только входные условия (пространство входов), но и пространство результатов.

Анализ причинно-следственных связей. Метод анализа причинно-следственных связей помогает системно выбирать высоко результативные тесты. Он дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций. Спецификация разбивается на «рабочие» участки. В спецификации определяются множество причин и множество следствий. Причина – это отдельное входное условие или класс эквивалентности входных условий. Следствие – это выходное условие или преобразование системы. Каждым причине и следствию приписывается отдельный номер. На основе анализа семантического содержания спецификации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин.

Предположение об ошибке. Основная идея метода состоит в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты.

Рассмотрим тестирование на примере привязки свойства к версии файла. Исходными данными для связи версии файла со свойством являются:

* ID версии файла;
* ID свойства;
* Значение свойства, которое мы хотим привязать.

При использовании метода эквивалентных разбиений, классами эквивалентности будут передаваемые параметры:

* ID версии файла – любые числа;
* ID свойства – такое целое число;
* Значение свойства – любая текстовая строка.

По методу анализа граничных условий, граничными условиями для исходных данных будут:

* ID версии файла. ID может быть только целым числом, следовательно, первой границей будет – целые и дробные числа. Второй границей является то, что ID версии файла может быть только таким числом, которое существует в таблице fileVersion.
* ID свойства. Аналогично ID версии, оно может быть только целым числом, следовательно, первой границей будет – целые и дробные числа. Второй границей является то, что ID может быть только таким числом, которое существует в таблице property.
* Значение свойства. У данного входного параметра отсутствуют граничные значения

Граничное значение для выходных данных: результатом работы может быть сообщение об ошибке при проверке правильности исходных данных или сообщение об успешном завершении работы алгоритма.

По методу анализа причинно-следственных связей можно выделить две причинно-следственные связи: одна – вывод пользователю сообщения об ошибке правильности исходных данных, вторая – вывод пользователю сообщения об успешном добавлении свойства к версии файла.

|  |  |  |
| --- | --- | --- |
| ID версии существует | ID свойства существует | Результат |
| 0 | 0 | Сообщение об ошибке |
| 1 | 0 | Сообщение об ошибке |
| 0 | 1 | Сообщение об ошибке |
| 1 | 1 | Сообщение о успешном завершении |

При тестировании, для обеспечения полной проверки исходных данных, необходимо проверить все 4 приведенных выше варианта.

**Оценочное тестирование**

После завершения комплексного тестирования приступают к оценочному тестированию. Целью данного тестирования является тестирование программы на соответствие основным требованиям. В рамках данного проекта были проведены следующе виды тестирования:

* тестирование удобства использования;
* тестирование на предельных объемах;
* тестирование защиты;
* тестирование удобства установки;
* тестирование совместимости;
* тестирование документации.

При проведении тестирования на удобство использования было установлено, что у пользователей, не имеющих прав модератора или администратора, в левом меню, которое занимает достаточно много места, слишком мало элементов. В результате, было принято решение для обычных пользователей сделать другой вид меню, который располагается сверху, во всю ширину страницы. Такое меню стало занимать намного меньше места, при этом, не потеряв в функциональности.

При проведении тестирования на предельных объемах было одновременно, одним пользователем, добавлено в каталог большое количество файлов. В результате, была обнаружена ошибка, которая заключалась в том, что после работы с базой данных, соединение с ней не закрывалось. Эта ошибка приводила к тому, что база данных выдала ошибку, сообщая о том, что достигнуто максимальное количество соединений.

После тестирования защиты было установлено, что любой пользователь может перейти на страницу просмотра категорий, которая должна быть доступна только модераторам и администраторам. Аналогичная ошибка была обнаружена на странице экспорта файла. Данная страница должна быть доступна только администраторам, но модераторы так же имели к ней доступ.

После проведения тестирования удобства установки были выявлено, что редактирование настроек соединения с базой данных и с Active Directory необходимо менять после каждой перезагрузки сервера. Это происходило потому, что файлы настроек лежали внутри архива приложения. Для решения этой проблемы было решено при запуске приложения проверять наличие файлов настроек в домашней директории пользователя и, если этих файлов нет, создавать их и предлагать пользователю настроить параметры соединения.

В рамках тестирования совместимости производился запуск разработанного приложения на различных операционных системах, а именно, на Windows 10, Mac OS X и Ubuntu 14.04. В результате данного тестирования было установлено, что на всех приведенных операционных системах приложение работает в нормальном режиме и никаких ошибок выявлено не было.

**Результаты тестирования**

В данном разделе было произведено тестирования разработанного приложения различными методами.