Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Южный Федеральный Университет»

Факультет математики, механики и компьютерных наук

Направление подготовки 010501 — «Прикладная математика и информатика»

Тестирование черного ящика на предмет анализа граничных значений

Выпускная квалификационная работа
на степень бакалавра
студента
Алексея Алексеевича Тактарова

Научный руководитель:
доцент, к.т.н.
Роман Ахмедович Хади

Ростов-на-Дону
2012

Содержание

Введение

Способность ошибаться, к сожалению, является неотъемлемой частью человеческой природы. Следовательно, любое изделие или продукт, произведенный человеком, может содержать ошибки, т.е. не соответствовать тем требованиям, которые были заложены в модель до изготовления. В эпоху ручного труда качество производства полностью определялось квалификацией мастера, что объясняло длительность и трудоемкость ремесленных процессов того времени. Развитие промышленности и появление конвейерного производства способствовали возникновению принципиально новых методов контроля качества, которые впоследствии легли в основу полноценной научной теории[1].

С появлением электронной вычислительной техники изменилась и форма продукта производства. Возникли такие отрасли, как программная инженерия, проектирование и разработка программного обеспечения, конечным результатом которых является компьютерная программа, т.е. информация.

Нельзя, тем не менее, считать, что области, связанные с разработкой программного обеспечения, свободны от человеческих ошибок. Классическим примером, который иллюстрирует роль контроля качества в процессе проектирования компьютерных программ и систем, является неудачный запуск космического аппарата “Маринер-1” в 1962 году: пропущенный дефис в программе бортового компьютера стал причиной крушения аппарата, что привело к провалу всего проекта стоимостью 20 миллионов долларов[**?**].

В настоящее время существуют разные методы оценки качества программного обеспечения, которые могут применяться в течение всего так называемого “жизненного цикла” проекта. Наиболее распространенным среди разработчиков является тестирование, в основе которого лежит проверка программы на соответствие заданным требованиям посредством ряда тестов. Практика тестирования программного обеспечения получила широкое распространение благодаря своей эффективно-

сти, а в некоторых методиках разработки даже стала предшествовать написанию кода программ (Test Driven Development или TDD — “разработка через тестирование”[**?**]).

Таким образом, тестирование программного обеспечения является актуальным средством контроля качества разработки, благодаря которому возможно создание высокоэффективных отказоустойчивых программных продуктов.

Объект, предмет и цель исследования

Данная работа посвящена проблеме контроля качества программного обеспечения, в частности, проблеме проверки конечного продукта разработки на соответствие поставленным требованиям и существующим стандартам в условиях отсутствия исходных текстов.

Предметом исследования является метод черного ящика для поиска ошибок в программном обеспечении.

Цель работы заключается в создании программного средства, позволяющего выполнять автоматическое стресс–тестирование программ с использованием метода черного ящика.

Были поставлены задачи, выполнение которых необходимо для достижения данной цели:

1. Рассмотрение современных методов контроля качества программного обеспечения.

2. Исследование существующих форм тестирования, выбор наиболее подходящей модели для практической реализации при заданных условиях.

3. Разработка программного продукта на основе выбранного метода.

4. Анализ полученных результатов.

1. Методы тестирования программного обеспечения

1.1. Верификация, валидация и тестирование

В основе теории, занимающейся оценкой качества программного обеспечения, лежат три понятия, которые на первый взгляд являются довольно близкими по значению: верификация, валидация и тестирование. В результаты схожести этих терминов, многие авторы интерпретируют их по-разному[?], что является недопустимым. Дадим развернутые определения каждого из понятий.

Верификация — это проверка соответствия результатов отдельных этапов разработки программной системы требованиям и ограничениям. Задача верификации — убедиться, что программное обеспечение, является функциональным, надежным, производительным, переносимым и отличается удобством сопровождения[?] [?].

Под валидацией понимают процесс проверки того, что требования заказчика(или пользователя) системы удовлетворены. Чаще всего валидация выполняется уже после разработки и рассматривает качество программного обеспечения с точки зрения конечного потребителя.

Наконец, тестирование является методом, при помощи которого валидация или верификация могут быть достигнуты. Однако иногда полная проверка соответствия на практике может быть невозможной, в таком случае задача тестирования — показать, что какое-то из требований не выполняется[?].

1.2. Эффективность тестирования при оценке качества

Рассмотрим основные технологии контроля качества программного обеспечения:

1. *Статический анализ исходного кода* используется при разработке

для выявления ошибок в текстах программы на некотором языке программирования. Для этого применяются: синтаксический разбор, поиск определенных шаблонов и подозрительных мест в коде. К сожалению, данный подход является нечуствительным к ошибкам времени выполнения, кроме того он оказывается малоэффективным при работе с языками, имеющими динамическую типизацию.

2. *Формальная верификация* представляет собой довольно трудозатратный метод проверки программы на соответствие некоторой математической модели. Различают дедуктивный и модельный способы верификации.

   Модельная верификация основывается на представлении программы в виде машины или автомата. Данный подход позволяет проводить не только полную, но и частичную верификацию которая может быть направлена на проверку только одного небольшого свойства, абстрагировавшись от менее важных деталей системы[?].

   При использовании дедуктивной верификации программа воспринимается как булевская функция многих переменных $f(x_1, x_2, ..., x_n)$, цель исследования при этом заключается в доказательстве, является ли данная функция выполнимой, т.е. существует ли такой набор данных $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$, что:

   $$f(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}) \equiv 1$$

   Разрешением данной проблемы занимается теория SAT (Satisfiability Modulo Theories), в которой применяются так называемые SAT–решатели, позволяющие выполнять доказательство, не прибегая к полному перебору.

   Исследованием задач подобного рода занимаются научные отделы крупных корпораций, для которых поддержание качества программного обеспечения имеет огромное значение. Примером может служить закрытая система под названием “SAGE”, разрабатываемая Microsoft Research для внутреннего использования[?].

3. *Тестирование* является методом, при котором о качестве продукта (объекта тестирования) можно судить по результатам конечного числа тестов — специально выбранных сценариев для проверки объекта. В отличие от статического анализа, тестирование может быть проведено на этапе выполнения программы(динамическое тестирование). Кроме того, в некоторых случая для проведения тестов необязательно знать о внутреннем устройстве программы (такой метод тестирования называется функциональным тестированием или тестированием черного ящика[**?**] [**?**]).

Таким образом, в условиях отсутствия исходных текстов тестирование является наиболее эффективным методом оценки качества программного обеспечения.

1.3. Модель тестирования

Центральной сущностью в тестировании является *объект*, т.е. программа. Объект тестирования помещается в окружение, которое определяется конкретным тестом. При этом выдвигается некоторое предположение о том, каким будет конечное состояние объекта (после прохождения теста). Совокупность этих предположений формирует *эталонную модель*. Настоящим же результатом теста может быть либо соответствие модели, либо некоторый *симптом* (рисунок 1).

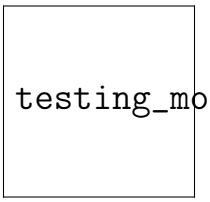

Рис. 1. Модель тестирования

Симптомом(или сбоем), согласно стандарту IEEE94, называют наблюдаемое аномальное поведение объекта, такое как несоответствие требованием или возникновение незапланированных явлений. Причиной симптома является *сбой*, т.е. некорректное состояние программы. Наконец, сбой является прямым следствие *ошибки* — просчета в проектировании программы[?] [?]. На рисунке 2 изображена взаимосвязь этих терминов. Таким образом, наблюдая симптом в результате тестирования, можно сделать предположение, что проявилась некоторая ошибка программы (именно предположение, поскольку в редких случаях симптом может возникнуть в результате сбоя операционной системы).

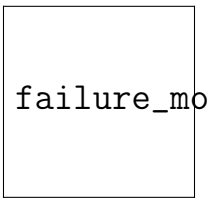

Рис. 2. Модель программной ошибки

1.4. Классификация видов тестирования

К сожалению, не существует универсального метода, который бы позволял тестировать программы одинаково эффективно. Выбор конкретного решения зависит от условий, в которых находится объект исследования. Например, наличие или отсутствие исходных текстов программы разбивает методы тестирования на следующие классы:

1. *Тестирование черного ящика* (англ. *black-box testing*) применяется, когда исходные тексты программы недоступны. В этом случае программа представляется в виде некоторой системы, имеющей входы и выходы. Манипулируя входными данными, тестировщик получает соответствующие данные на выходе, подтверждая или опровергая тем самым некоторые гипотезы.

2. *Тестирование белого ящика* (англ. *white-box testing*) соответственно применяется только при условии наличия исходных кодов. Проектирование тестовых ситуаций при этом происходит на основании внутреннего устройства и логики программы[**?**].

3. *Тестирование серого ящика* (англ. *grey-box testing*) можно назвать подвидом тестирования черного ящика с тем лишь исключением, что тестировщик имеет доступ к исходному коду, однако при непосредственном выполнении тестов исходный текст не требуется.

Исходя из модели тестирования (п. 1.3), результатом теста является соответствие требованию, либо некоторый симптом. Если желаемым итогом является проявление симптомов, то такое вид тестирования называется *стресс–тестированием*, а соответствующие тесты называются *“грязными”*.

В рамках данной работы ограничимся рассмотрением лишь методов стресс–тестирования черного ящика.

2. Тестирование черного ящика

2.1. Модель черного ящика

Термин “черный ящик” применительно к изучению систем, устройство которых неизвестно, ввел в употребление пионер в области кибернетики Уильям Росс Эшби[?]. Рассмотрим классическое определение.

Пусть X, Y — некоторые множества, называемые “множество входов” и “множество выходов” соответственно. Рассмотрим отображение $f : X \times T \to Y$, где T — временное множество (его можно определять как множество вещественных чисел, целых и т.д.).

Тогда четверка (X, Y, T, f) называется черным ящиком. Появление времени в данном определении неслучайно. Дело в том, что Эшби рассматривал такие системы, в которых одни и те же входные данные могли давать разный выходной результат. Изучение таких систем приводит к появлению вероятностных автоматов.

Назовем черный ящик детерминированным, если функция f не зависит от времени, т.е.:

$$\forall x \in X, \forall t \in T \quad f(x, t) = f(x)$$

Таким образом, детерминированный черный ящик есть тройка (X, Y, f). Эшби показал, что любой недетерминированный черный ящик может быть сведен к детерминированному, путем увеличения входного множества. Поэтому в дальнейшем под черным ящиком будем понимать лишь детерминированный черный ящик.

Схематически черный ящик можно представить в виде некоторого блока с подведенными входом и выходом (рисунок 3).

Иногда удобно рассматривать входные и выходные множества в виде декартова произведения некоторых множеств, т.е.:

$$X = X_1 \times X_2 \times ... \times X_n$$

$$Y = Y_1 \times Y_2 \times ... \times Y_m$$

Теперь черный ящик имеет n входов и m выходов (рисунок 4).

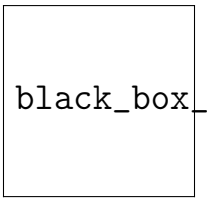

Рис. 3. Схематическое представление черного ящика

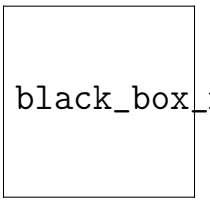

Рис. 4. Черный ящик с n входами и m выходами

2.2. Компьютерная программа как черный ящик

Работу компьютерной программу можно рассматривать как работу некоторого черного ящика: подан вход — получен результат. Проблема заключается в том, что необходимо каким-то образом определить, какими будут входное и выходное множества. Естественно было бы предположить, что входное множество — это множество всех данных, с которыми программа работает. Например, для программ, занимающихся обработкой текста, входное множество есть множество всех возможных файлов.

Однако, в общем случае данное предположение является неверным, поскольку компьютерная программа на практике не является детерминированной, т.е. имеет побочные эффекты. В этом случае к ящику следует добавить дополнительные входы и выходы, которые характеризуют состояние операционной системы, системное время, состояние всех устройств(в том числе сетевых), состояние всех компьютеров в локальной сети, состояние всех компьютеров в глобальной сети...

Учитывать все побочные эффекты довольно сложно и неэффективно, поскольку при этом метод черного ящика просто теряет смысл. Поэтому при тестировании вводятся некоторые допущения относительно поведения программы, что позволяет рассматривать ее в виде черного

ящика.

2.3. Анализ граничных значений

Поскольку перебор всех возможных входных значений черного ящика при тестировании чаще всего неосуществим, необходимы способы, с помощью которых можно провести анализ за конечное время. Можно заметить, что разные данные в некоторых случаях дают схожий результат, что приводит к предположению о том, что они обрабатываются с использованием одинаковой логики.

Пусть на входном множестве X введено некоторое отношение эквивалентности α. Тогда X разбивается на непересекающиеся классы эквивалентности:

$$[x]_\alpha = \{y \in X | x \alpha y\},$$

где x представитель класса. Эти условия порождают метод тестирования, который называется тестированием по разбиению. Перечислим гипотезы, на которых основывается данный метод:

1. Входные данные из одного класса эквивалентности приводят к идентичному поведению черного ящика.

2. Поведение на границах имеет наибольшие шансы быть некорректным.

Последнее означает, что границы классов являются потенциальным источником дефектов. Таким образом, проверка черного ящика сводится к проверке представителей классов и нескольких точек, близких к границам.

В случае, когда входы ящика являются числовыми, классы эквивалентности называются доменами[**?**]. Под доменом понимается область, которая может быть задана системой нелинейных (в общем случае) неравеств (рисунок 5). В этом смысле тестирование по разбиению становится похожим на решение задач нелинейного программирования [**?**].

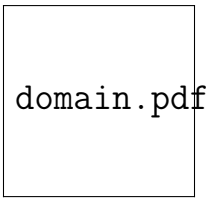

Рис. 5. Домен на плоскости. Точка A — представитель домена, точка B — граничная, точка C — в окрестности границы

2.4. Поиск границ доменов

Тестирование черного ящика усложняется, когда границы доменов неявно заданы, либо вообще неизвестны. С задачей поиска границы в этом случае хорошо справляются разнообразные стохастические методы[**?**].

Простейшим является поиск “наудачу”. При использовании этого алгоритма точки на входном множестве выбирается случайным образом до тех пор, пока применение их к черному ящику не вызывает симптомом (рисунок 6). Возникновение симптома означает с определенной долей вероятности, что точка расположена вблизи границы. В качестве подтверждения могут тестироваться соседние точки к данной.

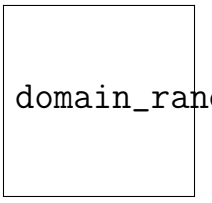

Рис. 6. Поиск границ случайным выбором точек. Красным отмечены точки, которые привели к появлению симптомов

На практике более эффективным может оказаться подход с использованием “мутации представителя”. Перед началом поиска выбирается точка, заведомо принадлежащая домену. Затем эта точка под влиянием “шума” искажается. Процесс повторяется для новой точки до тех пор, пока не проявляется симптом. Если за максимальное число шагов симптом

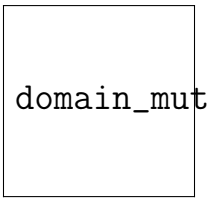

Рис. 7. Поиск границ с помощью “мутации представителя”. Точка A — представитель

не получен, мутация опять применяется к исходной точке (рисунок 7). Заметим, что способ мутации выбирается в зависимости от конкретной ситуации.

3. Средство автоматического стресс–тестирования программ

В рамках данной работы было разработано средство для автоматического стресс–тестирования программного обеспечения с использованием метода черного ящика.

В качестве входа выбрано множество файлов, имеющих преимущественно бинарный формат, поэтому средство применимо к довольно широкому классу программ (например, архиваторы, медиа–проигрыватели, компоненты операционной системы и т.д.).

3.1. Архитектура приложения

В состав средства тестирования входят три функциональных компонента: низкоуровневый отладчик, основной модуль и веб–интерфейс (рисунок 8).

Отладчик Задача отладчика — запуск программ на исполнения для выявления симптомов. В качестве профилирующего инструмента используется интерфейс отладчика операционной системы Windows (Win32 De-

bugger API [?]), он позволяет инспектировать возникающие в тестируемом приложении события и исключительные ситуации. Каждая запись о событии сериализуется в формат JSON[?], а затем передается следующему компоненту системы.

Отладчик реализован в виде программы на языке программирования C++; для работы с форматом JSON использована библиотека с открытым исходным кодом *json-cpp* [?].

Основной модуль Ядром приложения является модуль, который занимается подготовкой тестов (формированием входных файлов), запуском отладчика и анализом полученных результатов. Процесс запуска и профилирования объекта тестирования скрыт от этого компонента, поскольку он лишь использует JSON-интерфейс для доступа к событиям отладки. Накопленная статистика используется для дальнейшего отображения в веб–интерфейсе.

Данный компонент написан с использованием событийно–ориентированного фреймворка *node.js*[?], в качестве языка программирования был выбран скриптовый язык JavaScript. Этот выбор был обусловлен наличием в языке средств функционального программирования (это позволяло при разработке делать максимальный упор на алгоритм) и его минималистичность. Кроме того, язык обладает рефлексивностью, что делает его незаменимым при написании программ с возможностью добавления сценариев (одна из отличительных черт программ для тестирования).

Веб–интерфейс Накопленная в результате тестов статистика отображается в веб–интерфейс программы посредством асинхронных *http*–запросов, причем их обработка также происходит с помощью фреймворка *node.js*. Это возможно благодаря встроенному в *node.js* веб–серверу и дополнительным библиотекам[?].

Кроме того, была реализована возможность построения на стороне клиента потоковых карт пройденных тестов (рисунок 9). Потоковая кар-

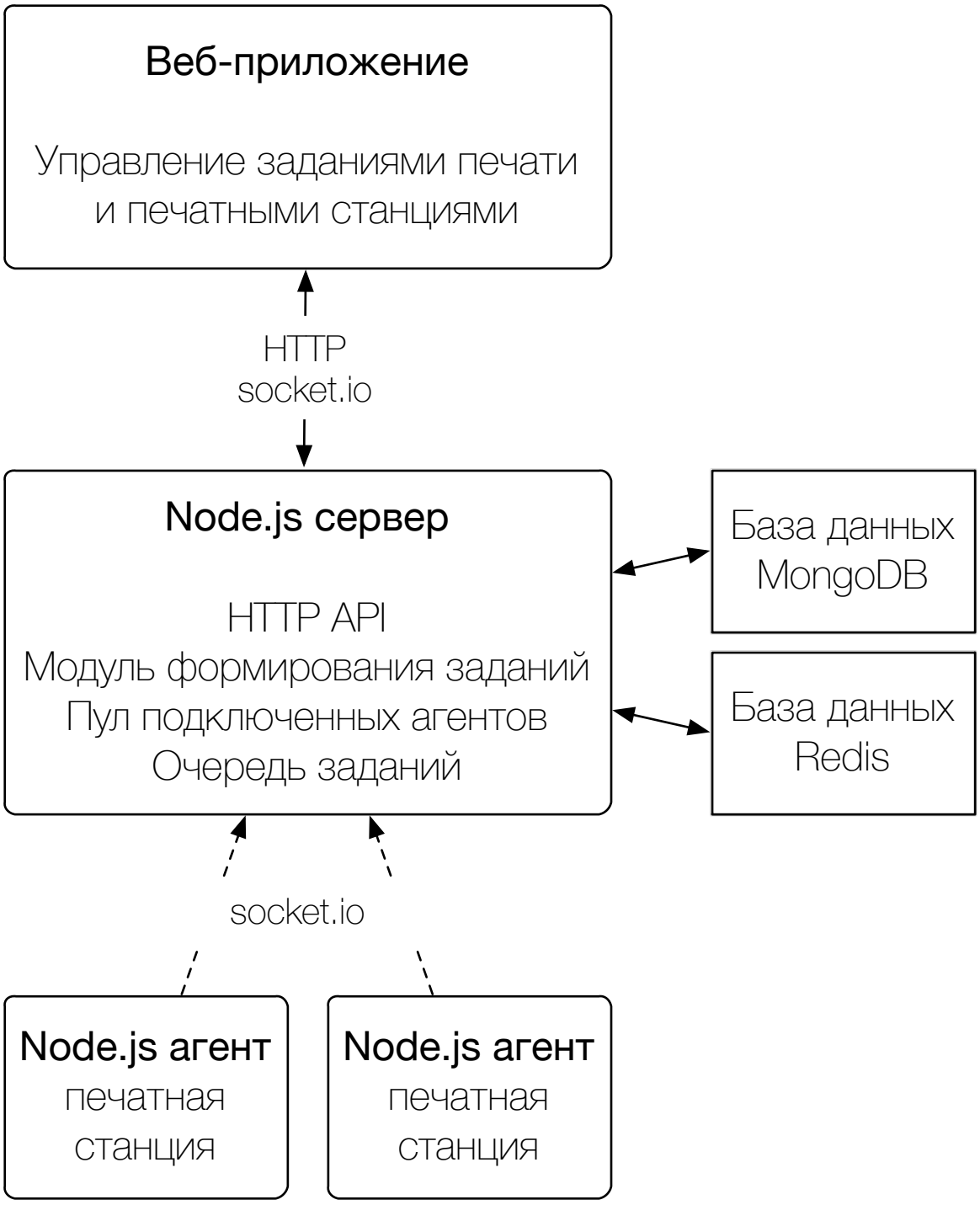

Рис. 8. Архитектура разработанного приложения

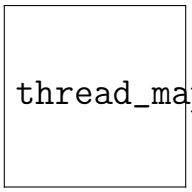

Рис. 9. Потоковая карта, построенная при помощи разработанного приложения

та — это диаграмма работы потоков программы во времени, с проявившимися симптомами. При помощи таких карт возможнен чисто визуальных анализ результатов конкретного теста.

3.2. Апробация методов

Для проверки работоспособности разработанного продукта был проведен ряд экспериментов, задачей которых был поиск ошибок некоторого программного обеспечения. Предприняты попытки поиска ошибок в следующих программах:

1. *WinDjView*[**?**] (просмотрщик djvu–документов) — применялся стохастический метод для генерации произвольного djvu–файла.

2. *Tesseract*[**?**] (программа для распознавания текста) — применялся стохастический метод для генерации произвольного jpeg–файла, кроме части файла, отвечающей за начало заголовка.

3. *Bmap*[**?**] (свободный просмотрщик изображений) — применялся мутационный метод на основе выбранного jpeg–изображения. В качестве мутации рассматривалось произвольное изменение частей заголовка файла.

4. *TinyImg*[**?**](свободный просмотрщик изображений, небольшой проект) — применялась аналогичная мутация заголовка исходного bmp–изображения.

В ходе последнего эксперимента была выявлена ошибка, проявляющаяся в отсутствии проверки на размер изображения в заголовке bmp-изображения.

Отсутствие найденных границ в первых трех случаях свидетельствует о соблюденных нормах контроля качества при разработке, что соответствует статусу проектов (разработка ведется на протяжении многих лет квалифицированными специалистами).

Заключение

В ходе проделанной работы были получены следующие результаты:

1. Разработано средство для автоматического стресс-тестирования программного обеспечения посредством входных файлов.

2. Реализованы стохастический и мутационный методы поиска граничных значений черного ящика.

3. При апробации средства были выявлены ошибки в одной из тестируемых программ, что подтверждает работоспособность реализованных методов.

Таким образом, тестирование является мощным средством для выявления ошибок в программном обеспечении, позволяя добиться стабильности, эффективности и надежного разрабатываемых продуктов.

Использование метода черного ящика делает возможным применение автоматизации при тестировании, что является альтернативой трудоемкого ручного поиска программных дефектов.

Список литературы

1. K. Ishikawa. *Guide to Quality Control*. Asian Productivity Organization, 1976.