*УДК 1004.415.53(075)*

# ИТЕРАЦИОННЫЙ ПОДХОД К ПОВЫШЕНИЮ КАЧЕСТВА ТЕСТИРОВАНИЯ ПРОГРАММ

**Б.М. Басок[1,@], к.т.н., доцент**
**В.Н. Захаров[2], д.т.н., ученый секретарь**
**С.Л. Френкель[2], к.т.н., с.н.с.**

*[1]Московский технологический университет (МИРЭА), Москва 119454, Россия*
*[2]Федеральный исследовательский центр «Информатика и управление» Российской академии наук (ФИЦ ИУ РАН), Москва 119333, Россия*
*[@]Автор для переписки, e-mail: basok@mirea.ru*

В статье рассмотрена задача повышения качества тестирования программных средств. Обобщение существующих методов синтеза тестов показало, что они не всегда гарантируют приемлемую полноту тестирования. Поэтому в настоящей работе предложен подход к разработке дополнительных тестов, который основан на анализе базы данных ошибок и возможностей программистов и тестеров. Он не требует знания программного кода и может использоваться при тестировании методом «черного ящика». Обсуждаются основные типы занесенных в базу данных ошибок, уровень их влияния на программный продукт, время, затраченное на их обнаружение и исправление. По результатам анализа ошибок выбираются функции программы, подлежащие дополнительному тестированию. Дополнительные тесты должны подтвердить или опровергнуть вывод о достаточной полноте тестирования функций программы традиционными методами. При выбранном подходе последовательность циклов системного тестирования программных средств рассматривается как итерационный процесс выявления возможных программных ошибок, в рамках которого построение новых сегментов тестов выполняется по результатам предыдущих циклов. Приведены некоторые практические результаты, полученные при использовании рассматриваемого подхода. Обсуждаются пути дальнейшего развития исследований в данном направлении.

***Ключевые слова:*** база данных ошибок, дополнительные тесты, цикл системного тестирования.

# ITERATIVE APPROACH TO INCREASING QUALITY OF PROGRAMS TESTING

**B.M. Basok[1,@],**
**V.N. Zakharov[2],**
**S.L. Frenkel[2]**

*[1]Moscow Technological University (MIREA), Moscow 119454, Russia*
*[2]Federal Research Center «Computer Science and Control» of the Russian Academy of Sciences, Moscow 119333, Russia*
*[@]Corresponding author e-mail: basok@mirea.ru*

The article deals with the problem of improving the quality of applied software (AS) testing. This testing begins with the pre-alpha version and ends at the stage of acceptance testing of the AS. This period includes the process of multiple execution of designed and developed system tests of AS, and includes additional analysis of the program in other ways. We refer to the testing process for a given period of time as a testing cycle (TC), which is usually implemented using the "black box" method without the tester's access to the program code. An analysis of existing methods of tests synthesis showed that these methods not always guarantee an acceptable completeness of testing. Therefore, in this paper we propose an approach to the development of additional tests, based on the analysis of the database of errors (DB) and the capabilities of programmers and testers. These additional tests should confirm or disprove the conclusions about the sufficient completeness of the testing of AS functions during the TC. The work analyzes the types of errors recorded in the database, the level of their influence on the AS, the time spent on their detection and correction. Based on the results of the error analysis, the AS functions that are subject to additional testing are selected. In this approach, the sequence of TC is considered as an iterative process of identifying possible program errors, in which new segments of tests are constructed according to the results of previous cycles. The article gives some practical results obtained in accordance with the proposed methodology and outlines new directions in the study of the effectiveness of AS testing.

***Keywords:*** the database of errors, additional tests, testing cycle.

## Введение

В настоящей статье обсуждается задача повышения качества тестирования собранных версий программных продуктов (ПП). Данное тестирование начинается с пред-альфа версии и заканчивается на этапе приемосдаточных испытаний ПП [1, 2]. Указанный период включает процесс многократного выполнения спроектированных и разработанных системных тестов ПП, а также дополнительный анализ программы другими способами: например, при локализации найденных отказов, при выявлении причин найденных отказов в данной функции, при тестировании методом свободного поиска (*exploratory testing*), когда тестирование осуществляется без заранее разработанных тестов. Будем называть процесс тестирования за данный период времени циклом системного тестирования (ЦСТ). ЦСТ обычно реализуется методом «черного ящика» без доступа тестера к программному коду, а последовательность указанных циклов – как *итерационный* процесс тестирования возможных программных ошибок, в рамках которого построение новых сегментов тестов выполняется по результатам предыдущих циклов.

В настоящее время предложены многочисленные подходы к разработке тестов программ [3–11]. Однако для современных сложных ПП, в разработке которых принимают участие десятки разработчиков, данные подходы не гарантируют приемлемой полноты тестирования. Это обусловлено тем, что разработка носит итерационный характер, требования к ней сложны и порой противоречивы, вследствие чего часто не соблюдаются требования контролепригодности ПП. Отсюда – неоправданно существенные затраты на тестирование ПП.

В работе дано краткое описание методики повышения эффективности существующих подходов, основанной на использовании информации об ошибках, накопленных на различных этапах процесса тестирования. Задача повышения уровня ЦСТ рассмотрена

с помощью построения дополнительных тестов, исходя из анализа базы данных (БД) ошибок и возможностей программистов и тестеров. Дополнительные тесты должны подтвердить или опровергнуть выводы о достаточной полноте тестирования функций ПП в течение ЦСТ. Кроме того, проанализированы основные типы возможных ошибок с точки зрения организации процесса тестирования и намечены пути новых направлений исследования проблемы эффективного тестирования программных продуктов.

## 1. Виды и классы программных ошибок

В процессе тестирования обнаруживаются ошибки разных классов. К ним относятся [1, 2]:

- ошибки при сборке версий;
- ошибки загрузки, инсталляции/деинсталляции, ошибки обновления версий программы;
- ошибки при установке программы в начальное состояние;
- ошибки в работе пользовательского интерфейса;
- функциональные ошибки;
- ошибки реакции на неверные данные, в частности, на неверные команды пользователя;
- ошибки взаимодействия с операционной средой;
- ошибки, связанные с ситуацией «гонок»;
- ошибки реакции на перегрузки;
- ошибки масштабирования;
- ошибки в системе безопасности;
- ошибки локализации.

По степени влияния на функционирование программы ошибки можно разделить на пять категорий:

- блокирующие («*Blocker*») – блокируют дальнейшее выполнение тестирования;
- критические («*Critical*») – аварийные остановки, потеря данных, серьезная «утечка» памяти;
- серьезные («*Major*») – значительные потери функциональности;
- незначительные («*Minor*») – небольшая потеря функциональности;
- тривиальные («*Trivial*») – незначительные проблемы, например, орфографические или стилистические ошибки.

Самой многочисленной группой ошибок являются ошибки категории *Minor*. К ошибкам этой категории можно отнести: ошибки инициализации программы, ошибки пользовательского интерфейса, ошибки при проверке границ данных при вводе, сброс значений заполненных полей при неправильной попытке регистрации, ошибки, связанные с неправильно выбранным направлением сортировок по некоему полю таблицы, и т.д.

Ошибок типа *Major* в программе меньше, чем ошибок типа *Minor*, однако их присутствие в программе оказывает важнейшее влияние на качество программы. Ошибок типа *Blocker* и *Critical* в программе очень немного, и они, как правило, выявляются на ранних стадиях тестирования. Ошибки типа *Trivial* легко находятся, исправляются, и к завершению ЦСТ они обычно отсутствуют.

## 2. Современные подходы к тестированию программ

Общий принцип разработки тестов основан на требовании, чтобы после нахождения и исправления найденных в течение ЦСТ ошибок убедиться, что количество оставшихся в ПП дефектов не существенно. Это необходимо сделать, поскольку стоимость локализации и исправления ошибки, найденной на этапе эксплуатации ПП, существенно выше, чем на этапе ее разработки. Так, по результатам независимого исследования ряда компаний [2, 3], соотношение указанной стоимости на соответствующих этапах составляет примерно 20:1. Кроме того, ошибки, найденные при эксплуатации программы, говорят о ее ненадежности и компрометируют коллектив разработчиков и компанию в целом. Поэтому важно обеспечить высокий уровень полноты ЦСТ и – в случае необходимости – определить группу подлежащих дополнительному тестированию функций. Здесь и далее под функциями понимаются все свойства программы, указанные в спецификации, удовлетворяющие стандартам и отвечающие ожиданиям пользователя. К ним можно отнести реализацию алгоритмов, стабильную работу (например, отсутствие утечки памяти и «гонок»), масштабирование, производительность и др.

Большинство из современных подходов основаны на использовании тех или иных структурно-функциональных связей в тестируемых программах. Например, в подходах, основанных на так называемых «контрактных спецификациях» [4, 5], программа представляется конечным автоматом, который строится по спецификации программы в виде описания предусловий и результатов операции каждой функции программы («контрактные спецификации»). При этом предусловия описывают область определения операции и косвенно отражают условия перехода в программе к рассматриваемой функции. Постусловие описывает ограничения на правильные результаты операции. Критерий качества тестирования – покрытие спецификаций. По данным авторов [4, 5], тесты, построенные таким образом, обнаружат в программе 70–75% возможных отказов. Это достаточно высокий уровень тестирования.

Известны и другие подходы, использующие автоматное представление программ [6]. Однако следует заметить, что сложные программные комплексы могут содержать тысячи и даже десятки тысяч ошибок. Поэтому абсолютная величина не выявленных таким методом ошибок будет весьма значительной. Кроме того, помимо очевидно высокой стоимости разработки подобных спецификаций (прежде всего, за счет высоких требований к квалификации тестеров, а также необходимости разработки дополнительных программных инструментов разработчика в случае построения спецификаций для достаточно сложных систем), существенной является проблема корректности спецификации, которая интенсивно изучается в последнее время [7]. Важно подчеркнуть, что во многих фирмах анализ спецификаций при построении тестов идет не формализовано, а интуитивно, и полнота покрытия возможных переходов программы из одних ее возможных состояний в другие также определяется интуитивными представлениями разработчика о возможных отказах и их последствиях.

Другим важным направлением разработки тестов ПП является синтез тестов относительно определенного класса отказов с дальнейшим анализом полноты этих тестов в выбранном классе отказов [8, 9] путем последовательного их внесения в исходный код ПП. Значимым примером такого подхода является мутационный подход и его модификации [10,

11]. Однако ввиду отсутствия доступа к коду на этапе ЦСТ мы не можем вносить искусственные дефекты в программу, чтобы тем самым, по крайней мере, оценить полноту тестов (полнота ЦСТ всегда будет ниже, ибо могут быть найдены ошибки, не обнаруженные при выполнении тестов). Таким образом, у нас отсутствует возможность построения дополнительных тестов, выявляющих непроверенные дефекты программы из заданного класса. По той же причине (отсутствие доступа к коду) мы не можем разрабатывать дополнительные тесты, ориентированные на покрытие незадействованных участков кода.

При тестировании программ на этапе ЦСТ разработчик тестов ориентируется только на покрытие функций (метод «черного ящика»). Однако даже при полном покрытии функций имеющимися тестами нет гарантии высокой полноты тестирования ввиду невозможности перебора большого количества данных, обрабатываемых этими функциями.

Известны подходы, основывающиеся на использовании аналитических динамических моделей надежности программ [12], которые позволяют на основании результатов выполнения тестов оценить и вероятность безотказной работы ПП за выбранный интервал времени, и примерную величину исходного количества отказов в программе, а, следовательно, приблизительное количество возможно оставшихся ошибок в программе. На основании полученных данных можно сделать выводы о необходимости разработки дополнительных тестов. Однако при этом не указываются места возникновения невыявленных отказов, то есть отсутствует информация для локализации выявленных ошибок и для построения дополнительных тестов.

Таким образом, краткий обзор методов разработки и использования тестов показал, что необходимо на этапе ЦСТ, исходя из высокой стоимости отказов, обнаруженных при эксплуатации ПП, и с целью обеспечения высокого уровня качества программ, разрабатывать подходы, позволяющие повысить уровень контроля ПП с помощью создания дополнительных программных тестов. Часто в этом случае полагаются на бета-тестирование, полагая, что пользователи найдут некоторые не выявленные в условиях обычного тестирования ошибки путем использования программы в рабочих режимах с различными данными [1, 2]. При этом покрывается большой объем кода. Действительно, такой прием достаточно эффективен, однако он также несовершенен. Не стоит забывать, что при бета-тестировании пользователь не имеет спецификаций и документации, часто отсутствует или несовершенна система помощи (*Help*). Потому бета-тестер, в основном, запускает рабочие примеры, с помощью которых осуществляется «позитивное» тестирование, в то время как задачей тестера является тестирование, связанное с созданием ситуаций, приводящих к неправильным результатам работы программы («негативное» тестирование). Этого можно достичь только с помощью специально подобранных данных, которые помогут выявить отказы, относящиеся к классам трудно проверяемых.

## 3. Организация процесса тестирования

Мы рассматриваем ЦСТ в условиях отсутствия формальной модели ошибок. Поэтому в качестве информации для построения тестов используются множества ранее обнаруженных ошибок, накопленных в БД, в которую включаются как исправленные, так и отложенные на будущее. Данная БД является обязательным элементом современных систем тестирования ПП любой степени сложности.

В самом общем виде предлагается следующая организация разработки тестов:

– разработчики программы выдают в группу тестирования спецификацию на тестирование, которая описывает основные функции программы, интерфейс пользователя, протоколы взаимодействия с интерфейсом и признаки их корректного выполнения, которые должны проверить тестировщики;

– тестеры разрабатывают тесты, выполняют необходимые прогоны тестируемых программ на полученных тестах, записывают обнаруженные ошибки в БД (при использовании соответствующих тестовых инструментов, в частности, это может выполняться автоматически), в которой для каждой ошибки указаны ее категория, статус (открыта, отклонена, исправлена, проверена и закрыта), ее краткое и подробное описание, функция, в которой обнаружена ошибка, имя теста, обнаружившего ошибку;

– после того, как по результатам тестирования ликвидировано подавляющее большинство найденных дефектов ПП, тестеры совместно с разработчиками принимают решение о необходимости разработки дополнительных тестов.

Ниже дается подробное описание действий разработчика тестов, реализующих итерационный процесс анализа результатов тестирования функций ПП, в котором итерации выполняются по последовательным этапам обнаружения ошибок в разработанных программах.

Имеются следующие итерации:

1. По завершении ЦСТ выделяются для дополнительного тестирования те функции, в которых выявлено наибольшее количество отказов. Поскольку у нас нет сведений о квалификации программистов и тестеров, а также о сложности отдельных функций, то можно предполагать наихудший вариант: либо функции с наибольшим количеством ошибок отличаются сложностью, либо квалификация реализующих их программистов оставляет желать лучшего. И то, и другое требует дополнительных проверок.

2. По завершении ЦСТ для дополнительного тестирования следует выделить те функции, в которых выявлено наименьшее количество отказов. Последнее свидетельствует или о высокой квалификации программистов, или о невысокой сложности проверяемых функций, или о низкой полноте тестов и невысокой квалификации тестеров. Если выбрать наихудший случай и предположить, что главная причина малого количества обнаруженных отказов в функции связана с тестами малой полноты, то данные функции нуждаются в дополнительном тестировании.

3. Если количество ошибок, обнаруженных тестами данной функции, существенно меньше количества ошибок, обнаруженных в результате ЦСТ тестами других функций, то тесты указанной функции невысокой полноты и нуждаются в доработке.

4. Нередко бывают случаи, когда тесты данной функции в течение ЦСТ неоднократно повторяются, причем количество повторных запусков существенно больше, чем у тестов других функций, по причинам поиска сложных, трудно проверяемых ошибок, обнаружения большого количества ошибок в функции. Очевидно, для подтверждения качества выполненного тестирования в таком случае желательны и дополнительные тесты указанной функции.

Следует заметить, что причиной многократного выполнения тестов также могут быть периодически появляющиеся и исчезающие «плавающие» ошибки. Обычно такие

ошибки связаны с асинхронным, параллельным выполнением функций с большим числом состояний (например, «гоночные» ошибки). В последнем случае помогают дополнительные проверки с использованием формальных методов, в частности, верификации моделей.

5. Если при тестировании функции среди обнаруженных ошибок много ошибок высокого уровня (*Major*) и выше, то это, по всей видимости, говорит о сложности задачи (алгоритма) и трудностях программирования. Нередко они обусловлены тем, что на ранних этапах разработки было выполнено некачественное тестирование и использовались (или не использовались вовсе) модульные тесты (*unit tests*) низкого качества. Не исключена некорректность или неполнота спецификации. Поэтому подобные функции подлежат дополнительному тестированию.

6. При анализе БД ошибок было обнаружено, что при регрессионном тестировании функции зафиксировано много неоднократно открываемых для исправления ошибок (статус *Reopen*). Если предположить, что программу проверяют тестеры примерно одной квалификации, то там, где ошибки чаще открывались повторно, программисты некачественно исправляли ошибки (и потому нет гарантий, что они качественно написали программу), либо возникли существенные трудности при нахождении данной ошибки. Значит, функция оказалась достаточно сложной. Исходя из сказанного, можно сделать вывод о желательности дополнительной проверки подобных функций.

7. Если в БД присутствует много отклоненных ошибок в какой-то функции, т.е. дефектов, найденных тестером ошибочно, то не исключено, что тестер недостаточно разобрался в работе функции. Поэтому такие функции также подлежат проверке.

Таким образом, целесообразно предложить следующий алгоритм разработки дополнительных тестов:

I. Определить по завершении выполнения всех тестов и исправления ошибок примерное количество оставшихся ошибок в программе с использованием аналитических динамических моделей надежности [12];

II. Если количество невыявленных отказов значительно, следует перейти к разработке дополнительных тестов в соответствии с описанной выше методикой, а затем перейти к повторному выполнению п. I;

III. Если количество невыявленных отказов приемлемо, то процесс дополнительного тестирования можно заканчивать.

Следует отметить, что решение о необходимости новой итерации ЦСТ можно принимать не только исходя из оценки оставшихся непроверенными отказов. Выводы о необходимости дальнейшего тестирования можно делать и на основании сравнительного анализа обнаруженных дополнительными тестами дефектов на данной и предыдущей итерациях, их распределения по типам и категориям ошибок.

## 4. Некоторые результаты

В качестве конкретного примера применения описанной в статье методики можно привести данные тестирования информационной программной системы, реализованной с использованием Интернет-технологий «*My Life*» («Моя жизнь»), предназначенной для описания биографии пользователя с фото-, аудио- и видео-вставками (оригинальная раз-

работка фирмы, в которой один из авторов статьи работал руководителем группы тестирования). Для проверки функционирования данного приложения было разработано порядка 40 тестов, содержащих от 20 до 50 входных наборов и ожидаемых реакций программы на них. Для автоматизации тестирования указанного приложения была создана библиотека тестовых скриптов, содержащая более 6000 строк. В качестве стенда тестирования использовалась система виртуальных машин VMWare GSX Server.

При системном автоматизированном тестировании вышеуказанного приложения методом «черного ящика» с использованием инструментальных средств фирм IBM Rational и Microsoft Corporation было обнаружено около 1500 ошибок, из которых примерно 75% составляли ошибки типа «*Minor*», 10% – «*Major*», 15% – «*Trivial*» и порядка десятка ошибок типа «*Blocker*» и «*Critical*». Все обнаруженные при тестировании отказы заносились в БД ошибок Jira компании Atlassian.

После ликвидации практически всех обнаруженных ошибок по описанной выше методике была разработана группа из 8 тестов. В результате удалось обнаружить еще порядка 10 дефектов, из которых один относился к «*Major*», остальные – типа «*Minor*». При это среди найденных ошибок «*Minor*» хотелось бы выделить ошибку, связанную с отсутствием перед записью на внешний носитель наличия у него необходимой для этого свободной памяти, а также ошибку, связанную с возможностью влияния пользователя на содержимое одной из страниц сайта в режиме *View* (Просмотр).

Полученные дефекты были занесены в БД ошибок и ликвидированы разработчиками приложения. После этого по данной методике с учетом новой информации в БД разработано еще несколько дополнительных тестов, с помощью которых удалось найти еще два отказа типа «*Minor*». На этом по согласованию с разработчиками приложения ЦСТ силами тестеров фирмы был закончен. Затем руководство фирмы передало приложение независимому тестовому агентству, которое, используя собственные дополнительные тесты, нашло еще две ошибки: «*Minor*» и «*Trivial*». После этого приложение поступило в опытную эксплуатацию.

В общей сложности, данный подход применялся при тестировании десяти программных комплексов, содержащих web-приложения. Количество найденных ошибок в программе до его применения составляло от 1000 до 10000. Тесты, разработанные в соответствии с рассмотренной методикой, позволили обнаружить в каждом комплексе дополнительно ряд дефектов различных типов и категорий и тем самым повысить эффективность тестирования и, соответственно, качество программ.

## Заключение

В статье рассмотрена методика выбора дополнительных тестов, обеспечивающих высокий уровень качества тестирования ПП. Эта методика не требует знания программного кода и, в первую очередь, может использоваться при тестировании методом «черного ящика». Данный подход основан на анализе БД ошибок, содержащих сведения о самых разнообразных отказах ПП. Дальнейшее развитие работ в этом направлении может быть связано с построением статистической модели, использующей содержимое БД ошибок для автоматического выбора функций программы, подлежащих дополнительному тестированию. С целью решения проблемы снижения стоимости тестирования будут приняты во внимание возможности применения методов Машинного Обучения (*Machine*

*Learning*), в рамках которого результаты тестирования на предыдущих ЦСТ рассматриваются как «обучающая» (*training*) выборка для выбора стратегии построения теста на следующей. Представляется перспективным также изучение возможности использования так называемых статистических баз данных, в которых поиск и извлечение информации выполняется по некоторому подмножеству данных, гарантирующему требуемую вероятность нахождения данных искомых.

*Работа выполнена при частичной финансовой поддержке РФФИ (проекты 16-07-01028 и 15-07-053160).*

**Литература:**

1. Басок Б.М., Красовский В.Е. Тестирование программного обеспечения. М.: МИРЭА, 2010. 120 с.

2. Куликов С.А. Тестирование программного обеспечения. Базовый курс. [Электронный ресурс]. – URL: http://svyatoslav.biz/software_testing_book/ (дата обращения: 26.06.2017).

3. Технология разработки программного обеспечения // Журнал информационных технологий. IT-технологии. С. 16. [Электронный ресурс]. URL: http://www.irkinfo.ru/tekhnologiya-razrabotki- programmnogo-obespecheniya-str16.html / (дата обращения: 28.06.2017).

4. Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов // Программирование. 2003. № 6. С. 25–43.

5. Петренко А.К., Кулямин В.В., Хорошилов А.В. Об интеграции формальных методов в задачах верификации операционных систем // Труды ИСП РАН. 2015. Т. 27. Вып. 5. С. 175–190.

6. Шалыто А.А., Туккель Н.И. Switch-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. 2001. № 5. С. 46–62.

7. Landsberg D., Chockler H., Kroening D., Lewis M. Evaluation of measures for statistical fault localisation and an optimising scheme // Fundamental Approaches to Software Engineering : 18th Int. Conf., FASE 2015. 2015/3/31. P. 115–129. [Электронный ресурс]. URL: Evaluation of Measures for Statistical Fault Localisation and an Optimising Scheme http://dblp.uni-trier.de/db/conf/fase/fase2015 (дата обращения: 28.06.2017).

8. Басок Б.М., Гречин А.А. Об усовершенствовании статистического метода оценки полноты тестов программ и устройств // Инструменты и методы анализа программ: Труды Междунар. науч.-практ. конф. Кострома, 2013. С. 40–45.

9. Wei J., Thomas A., Li G., Pattabiraman K. Quantifying the accuracy of high-level fault injection techniques for hardware faults / In: 44th IEEE/IFIP Int. Conf. on Dep. Sys. & Netw. (DSN ’14). IEEE, June 13, 2014. P. 375–382.

10. Harman M., Jia Y., Langdon W.B. A Manifesto for higher order mutation testing // Third Int. Conf. on Software Testing, Verification, and Validation Workshops (ICSTW), 2010. April 6–10, 2010. P. 80–89.

11. Басок Б.М., Гречин А.А. О едином подходе при анализе тестов дискретных устройств и программ // Вопросы радиоэлектроники. Серия ЭВТ. 2010. Вып. 3. С. 140–145.

12. Азовцев В.В. Исследование методов оценки и повышения надежности программного обеспечения. [Электронный ресурс]. – URL: http://www.azovikdip.ru / (дата обращения: 16.06.2017).