МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт энергетики и электротехники
(наименование института полностью)
Кафедра «Промышленная электроника»
(наименование кафедры)
11.04.04 Электроника и наноэлектроника
(код и наименование направления подготовки, специальности)
Электронные приборы и устройства
(направленность (профиль)/специализация)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Разработка метода тестирования электронных устройств с пользовательским интерфейсом»

Студент	Я. О. Герасимов (И.О. Фамилия)	(личная подпись)
Научный руководитель	к.т.н., Е. С. Глибин (ученая степень, звание, И.О. Фамилия)	(личная подпись)
Руководитель программы	д.т.н., профессор В.В. Ивашин (ученая степень, звание, И.О. Фамилия)	(личная подпись)

«____»_____2018г.

Допустить к защите

Заведующий кафедрой	к.т.н., доцент А.А. Шевцов (ученая степень, звание, И.О. Фамилия)	(личная подпись)

«____»_____2018г.

Тольятти 2018

Содержание

Введение

В качестве одной из важнейших высокотехнологичных отраслей промышленности в настоящее время производство полупроводников сталкивается с множеством возможностей и проблем одновременно. Быстрый рост полупроводниковой промышленности приносит большие экономические выгоды. Между тем, жесткая конкуренция требует, чтобы полупроводниковая компания повышала производительность и эффективно использовала ресурсы.

Очень важно иметь возможность протестировать и отладить своё устройство до его запуска в массовое производство. При этом все найденные ошибки и неточности нужно правильно зарегистрировать и проследить за их жизненным циклом. Для этого нам нужны специальные системы, которые бы значительно упрощали работу с уже найденными ошибками.

В данной диссертации изучены и разработаны новые методики для диагностирования и нахождения неисправностей или ошибок программного кода в управляемых полупроводниковых устройствах.

Область исследования

Полупроводниковые управляемые устройства.

Объект исследования

Методы тестирования управляемых полупроводниковых устройств. Системы отслеживания неисправностей и ошибок программного обеспечения.

Предмет исследования

Нахождение местоположения неисправности и ошибок программного обеспечения в управляемых полупроводниковых устройствах. Регистрация и наблюдение за ошибкой или неисправностью в разработанной системе отслеживания ошибок программного обеспечения.

Цель исследования

Разработка методов нахождения местоположения неисправности в управляемых полупроводниковых устройствах. Разработка системы отслеживания ошибок программного обеспечения.

Задачи исследования:

1. Анализ типовых проблем в управляемых полупроводниковых устройствах;
2. Разработка и исследование методологических рекомендаций по выявлению местоположения неисправностей или ошибок работы программного обеспечения в управляемых полупроводниковых устройствах;
3. Разработка и исследование систем для отслеживания неисправностей и ошибок программного обеспечения в управляемых полупроводниковых устройствах.

Методы исследования

Изучение и анализ текущих достижений в заданной области. Поиск возможных улучшений.

Научная новизна

Были разработаны и предложены новые методы анализа и тестирования управляемых полупроводниковых устройств. Составлены четкие требования, которым должна соответствовать система для отслеживания неисправностей и ошибок программного обеспечения в управляемых полупроводниковых устройствах.

Практическая значимость

Новые методы анализа и тестирования управляемых полупроводниковых устройств могут существенно упростить работу по отладке и ремонта полупроводниковых устройств. Занесение всех найденных проблем в систему отслеживания значительно улучшит продуктивность работы.

Апробация работы

Две публикации в сборнике научно-практической конференции «Студенческие Дни науки в ТГУ»

1. Аппарат для тестирования программ полупроводниковых цепей и текущие проблемы тестирования

Проектирование, тестирование и изготовление полупроводниковых схем, таких как микропроцессоры и микроконтроллеры, часто является дорогостоящим и трудоемким процессом. Для облегчения проектирования и изготовления интегральных схем были разработаны несколько типов автоматизированного проектирования. Как правило, создается эмуляция тестируемой схемы, и производятся различные тесты для выявления потенциальных проблем. Но программы для таких эмуляций тоже должны быть протестированы и отлажены, чтобы гарантировать, что ошибки программного обеспечения не влияют на работу эмулированной полупроводниковой схемы. Кроме того, каждое изменение существующей программы должно быть проверено, чтобы убедиться, что это изменение не влияет на работу исходного кода. Ввиду высоких затрат на разработку, связанных с производством полупроводников, крайне важно, чтобы ошибки программного обеспечения были обнаружены и исправлены в начале цикла разработки.

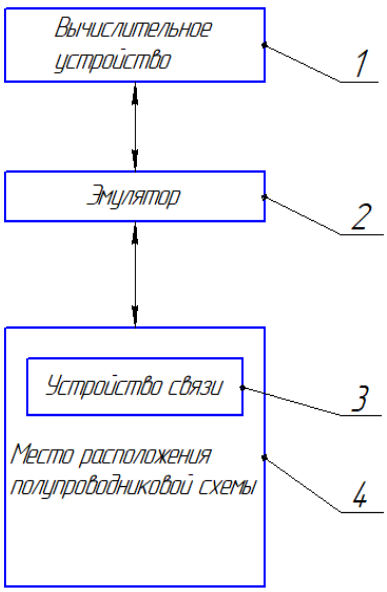

Рисунок 1 – Схема тестовой среды для полупроводниковых приборов.

На Рисунке 1 показана схема тестовой среды для полупроводниковых приборов. Давайте рассмотрим её подробнее. Программа, подлежащая тестированию, первоначально загружается в вычислительное устройство 1. Вычислительное устройство подключено к эмулятору 2 и устройству связи 3. Место расположения полупроводниковой схемы 4, как следует из названия, содержит тестируемую полупроводниковую схему. Устройство 3 связи представляет собой специальную версию сконструированной реальной интегральной схемы, где дополнительные соединения присоединены к внутренним цепям устройства для обеспечения доступа и управления устройством от внешних схем. Вычислительное устройство 1 управляет предварительно загруженной тестовой программой. Эмулятор 2 включает в себя логику захвата и управления, которая контролирует содержимое регистра адреса программы и внутреннюю шину данных и различные линии управления устройства связи 3 и передает захваченные данные в вычислительное устройство 1 для анализа. Устройство 3 связи обычно физически подключается к устройству, с которым в конечном итоге будет работать микроконтроллер.

Для каждого типа микропроцессора, требуется уникальное устройство связи. Кроме того, если изменяется скорость работы микропроцессора или изменяется размер памяти во время процесса разработки, требуется новое устройство связи. Таким образом, программное обеспечение для конкретного микропроцессора не может быть полностью протестировано до тех пор, пока не будет доступно точное и подходящее устройство связи. Аппаратное обеспечение, стороннее по отношению к устройству связи, быстро становится очень сложным, поскольку добавляются дополнительные функции отладки. Таким образом, значительное количество оборудования должно быть подключено к целевой плате, что приводит к механической и электрической ненадежности.

Когда программное обеспечение доставляется клиенту, вероятность того, что клиент обнаружит ошибку в программе, напрямую связана с процентом кода, который был выполнен во время тестирования. Это называется «охват кода». Например, тестирование калькулятора путем вычисления 2+2 не будет оценивать точность с плавающей запятой или тригонометрические возможности калькулятора. Однако, если весь функционал был протестирован исчерпывающе, клиент никогда не найдет ошибку. Конечно, тест, который умножит все возможные числа между собой и проверит результат, не является практическим решением.

Как правило, тесты должны максимально использовать программный код. В идеале каждый байт программного кода должен быть запущен, а результат проверен. Кроме того, необходимо определить и сообщить объем тестируемого кода относительно общего размера кода.

Существующие методы покрытия кода имеют ряд недостатков, которые в случае их преодоления могут расширить возможности инструментов тестирования программного обеспечения и точность базовых программ, проверенных такими инструментами. В частности, существующие инструменты охвата обычно связывают тестируемую программу с специальной библиотекой покрытия, тем самым увеличивая размер кода, предотвращая выполнение в режиме реального времени и не проверяя фактический отправленный код. По сути, результаты тестов получаются из программы, которая больше, медленнее и потенциально демонстрирует другое поведение, не такое, как у программы, которая будет запущена у клиентов. Кроме того, большинство существующих инструментов охвата работают, оценивая исходный код и не могут давать результаты тестирования, которые включают в себя покрытие файлов библиотеки, связанных с исходным кодом, например, как стандартные библиотеки C (для которых исходный код обычно недоступен). Наконец, большинство существующих инструментов охвата плохо интегрированы с исходным

кодом и выдают только частичные диаграммы, иллюстрирующие неиспользуемые пути, а не напрямую связывают результаты с исходным кодом.

В дополнение к важности тщательного тестирования программного обеспечения крайне важно записывать информацию о выполненных тестах, а также результаты тестов для документирования, проверки и аутентификации.

Исторически сложилось так, что разрозненные и зачастую несовместимые системы тестирования и отладки программного обеспечения затрудняли интеграцию тестовой информации в стандартную среду документации, проверки и аутентификации. Кроме того, обычные системы тестирования и отладки программного обеспечения обычно представляют собой автономные устройства, адаптированные к конкретному приложению. Таким образом, отсутствие стандартного интерфейса не позволяет пользователю контролировать процессы тестирования и отладки из другой прикладной программы.

Как видно из вышеописанных недостатков, существует потребность в недорогой системе для тестирования и отладки программного обеспечения программистом во время компиляции. Так же есть потребность в системе тестирования и отладки, которая точно сообщает время, которое было затрачено на выполнение программы, учитывая аппаратную конфигурацию и периферийные устройства, что позволит настраивать различные алгоритмы. Существует и спрос на системы тестирования и отладки программного обеспечения, которые бы могли создавать неограниченное количество контрольных точек при любых условиях. Ощущается недостаток и в инструменте покрытия кода, который бы работал на уровне машинного кода, не меняя тестируемую программу и интегрируя с инструментами отладки, чтобы привести пользователя непосредственно к соответствующему неиспользуемому исходному коду.

1.1 Что такое тест-кейс?

Тест-кейс (TestCase) – это описание алгоритма действий, которые необходимы для тестирования различного функционала программного обеспечения и не только. Тест-кейс является одним из главных документов для инженера по качеству, так как алгоритмы, написанные в кейсах, направлены на проверку различного функционала и помогают достигнуть фактического результата. Тест-кейсы объединяются в так называемые тест комплекты. Так же, важным документом в арсенале инженера по качеству является тест-план. В тест-плане мы описываем объем различных работ по тестированию, а так же отведенное на это время. Стоит понимать различия между тест-комплектом и тест-планом.

1.2 Зачем нужны тест-кейсы?

Тест-кейсы необходимы для проверки того или иного функционала, если мы не можем ознакомиться со всей проектной документацией. Грамотный и актуальный тест-кейс позволяет существенно экономить время и силы инженеров по качеству.

1.2.1 Атрибуты тест-кейса

Абсолютно все тест-кейсы должны иметь следующие атрибуты:

- Уникальный идентификатор тест-кейса;
- Название;
- Предусловия;
- Шаги;
- Ожидаемый результат.

Давайте рассмотрим каждый атрибут подробнее.

Уникальный идентификатор тест-кейса — обязателен для навигации по нашим тест-наборам и для организации хранения.

Название — основная идея тест-кейса. По сути, является его кратким описанием.

Предусловия — условия, которые необходимо выполнить перед непосредственным прохождением тест-кейса. Прямого отношения к функционалу, что мы проверяем не имеет.

Шаги — последовательность действий, которую мы выполняем для получения фактического результата.

Ожидаемый результат — результат выполнения шагов тест-кейса, который мы ожидаем получить. В идеале совпадает с фактическим результатом.

1.2.2 Важные пункты тест-кейса

Выполнение тест-кейса делится на три возможных результата:

1. Положительный результат, если фактический результат равен ожидаемому результату;
2. Отрицательный результат, если фактический результат не равен ожидаемому результату. В этом случае, найдена ошибка;
3. Выполнение теста блокировано, если после одного из шагов продолжение теста невозможно. В этом случае так же, найдена ошибка.

1.2.3 Нежелательные элементы тест-кейса

Здесь будут приведены пункты, которые стоит избегать при написании тест-кейса:

1. Зависимость от других тест-кейсов;

2. Неточная, размытая формулировка шагов или ожидаемого результата тест-кейса;

3. Недостаток нужной информации;

4. Чрезмерная детализация.

Рассмотрим, почему же эти элементы нежелательны в тест-кейсе.

Зависимость от других тест-кейсов — тест-кейс на который мы ссылаемся может устареть или вообще быть удаленным.

Неточная, размытая формулировка шагов или ожидаемого результата тест-кейса — не позволит инженеру по качеству выполнить тест-кейс.

Недостаток нужной информации — без наличия, например, пароля для входа в тестируемую систему, инженер по качеству не сможет выполнить тест-кейс.

Чрезмерная детализация — наличие лишней информации будет только рассеивать внимание инженера по качеству и не даст сосредоточиться на реально важных аспектах теста.

2. Системы отслеживания неисправностей и ошибок программного обеспечения

Системы отслеживания неисправностей и ошибок программного обеспечения являются важной частью разработки программного обеспечения. Они оказывают огромное влияние на эффективность использования системы. Информация, которая предоставляется об условиях возникновения ошибок, а так же о причинах их возникновения, помогает разработчикам программного обеспечения реагировать быстро и быть уверенными, что ошибки программного кода либо исправлены, либо исключены из системы. Большое количество информации, предоставленной в отчетах об ошибках программного кода, может вызвать проблемы у разработчиков при определении причин возникновения ошибок программного кода. Поэтому системы отслеживания ошибок должны постоянно улучшаться и следовать определенным стандартам. Для преодоления этой проблемы, предлагаются четыре фундаментальных направления для повышения эффективности систем отслеживания ошибок программного кода.

Система отслеживания неисправностей и ошибок программного обеспечения незаменима для любой системы, которая должна хорошо работать. Это инструмент, который ускоряет исправление ошибок и обеспечивает необходимое качество разрабатываемого или используемого программного обеспечения. Эти системы широко используются и рассматриваются как важные репозитории, которые помогают быстро определить статус ошибок и незамедлительно их решить. Таким образом, системы отслеживания неисправностей и ошибок программного обеспечения так же можно использовать для отслеживания прогресса проекта. Разработчики программного обеспечения часто используют отчеты об ошибках программного кода и пробуют исправить их, если недостаточно

информации было предоставлено. В опросе, проведенном для разработчиков программного обеспечения таких компаний, как Mozilla, Eclipse и Apache, было обнаружено, что информационные элементы, представленные в отчетах об ошибках, сыграли очень важную роль в исправлении ошибок программного обеспечения. Недостаточные или неправильные отчеты вызвали задержку в исправлении ошибок программного обеспечения и, таким образом, привели к пересечению дедлайнов. Информационные элементы, которые можно найти в отчетах об ошибках, включают скриншоты, тестовые примеры, ожидаемое поведение, наблюдаемое поведение, стек трейсы и шаги для воспроизведения. Как правило, это минимальная предпочтительная информация, необходимая инженерам для исправления ошибок программного обеспечения. Однако предыдущие исследования в этой области показали, что репортеры ошибок не указали эти важные информационные поля в своих отчетах об ошибках, что сделало их отчеты малоинформативными. Такие отчеты малополезны для разработчиков при исправлении ошибок. Когда разработчикам нужна информация очень подробная, а в отчетах об ошибках такая информация отсутствует, это приводит к остановке проекта или задержке с другими причинами и последствиями. Основная проблема с текущими системами отслеживания ошибок заключается в том, что они просто хранят некоторые информационные поля в базах данных и не описывают их характер. Предлагаемые методы в этом документе решают эту проблему, предоставляя надежные и практичные шаги или направления, которые могут практически улучшить понятность информации, а также обеспечить, предоставление и сохранность основных данных в отчетах об ошибках.

3. Жизненный цикл ошибки в системе отслеживания неисправностей и ошибок программного обеспечения

Разработка программного обеспечения сопряжена со многими проблемами. Одной из таких проблем является процесс отслеживания ошибок. Ошибки в процессе разработки программного обеспечения неизбежны. Однако они должны тщательно отслеживаться и фиксироваться, иначе они проявляются в конечном продукте и приводят к сбою в работе программного обеспечения. Ошибки программного обеспечения могут быть названы как «баг». Жизненный цикл разработки системы имеет много этапов. Команда разработчиков программного обеспечения может совершать ошибки на любом этапе. Однако, после написания кода, все ошибки всплывают на поверхность при первом же выполнение тестирования. Большинство ошибок программного обеспечения можно избежать путем правильного анализа и проектирования в соответствии с требованиями заказчика. Ошибки в программном коде приводят к сбою системы. Это означает, что такая система не имеет надлежащего качества и не соответствует ожиданиям клиента. Когда система может удовлетворить функциональные требования, предъявляемые клиентом, который считается качественным продуктом. Отслеживание ошибок имеет много фаз и имеет свой собственный жизненный цикл, как показано на рис. 2.

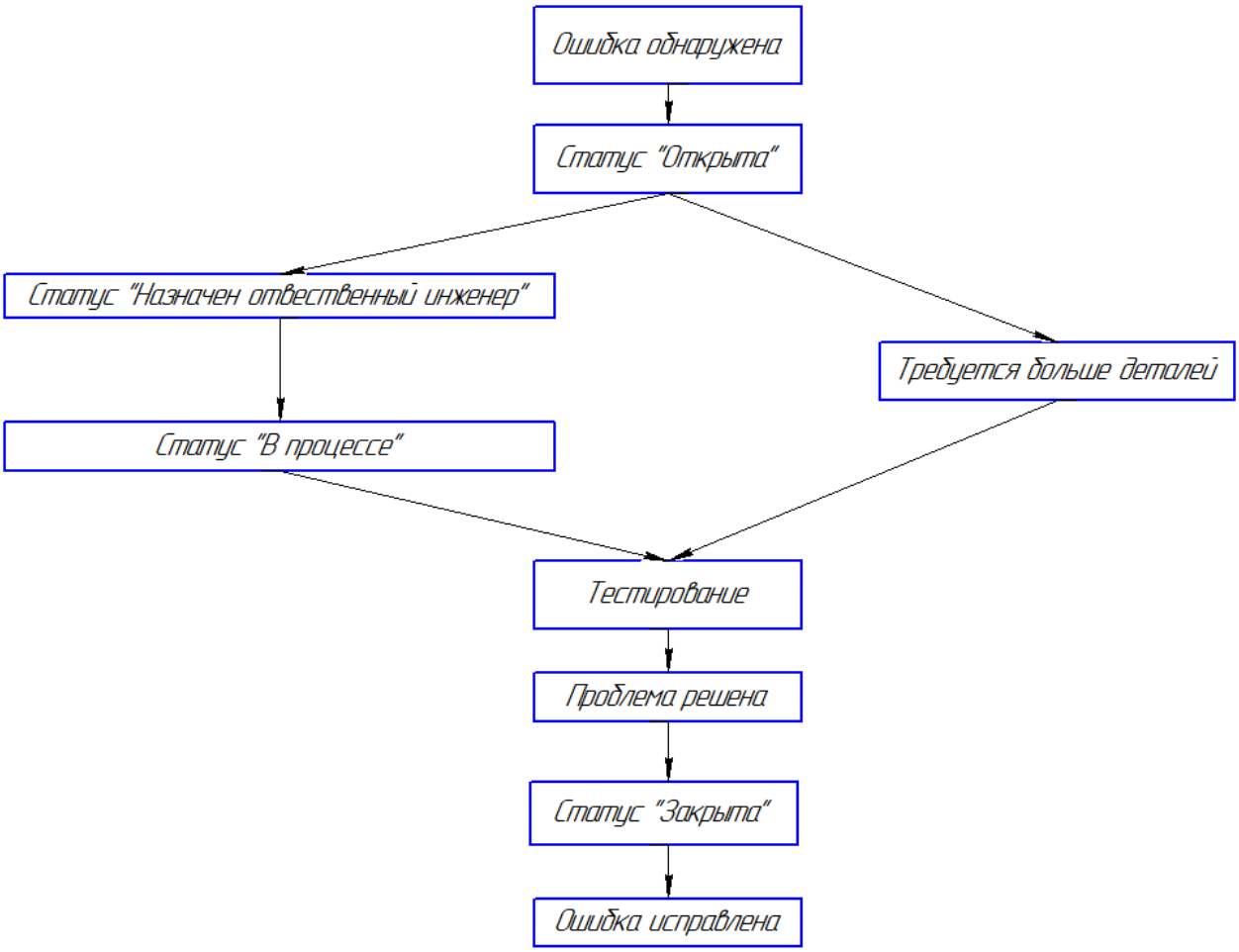

Рисунок 2 – Жизненный цикл ошибки в системе отслеживания неисправностей и ошибок программного обеспечения

Как можно видеть на рис. 2, отслеживание ошибок циклично по своей природе. С самого начала, до тех пор, пока ошибка не будет исправлена, она имеет свой жизненный цикл с определенными этапами. Поскольку ошибки заставляют систему работать неправильно, системы отслеживания ошибок программного обеспечения уменьшают вероятность наличия «багов» в системе долгое время. Команда разработчиков программного обеспечения может быстро исправить ошибки и закрыть их с помощью подходящей системы отслеживания ошибок. Отслеживание ошибок улучшает качество разрабатываемого программного обеспечения. Даже после выдачи программного обеспечения заказчику, могут быть обнаружены ошибки, и все та же система отслеживания будет использоваться для быстрого исправления этих ошибок.

4. Критерии и обзор существующих систем отслеживания неисправностей и ошибок программного обеспечения

Когда разрабатывается новое программное обеспечение, отслеживание ошибок является важной, а также сложной задачей. Традиционно отслеживание ошибок упрощается. На протяжении многих лет разработчики программного обеспечения использовали электронные таблицы для хранения сведений об ошибках и их исправления. Однако использование программного обеспечения, такого как электронная таблица, неэффективно, учитывая современные возможности, такие как электронная почта; RSS-ленты и уведомления через SMS и т. д. Тем не менее, очень полезно использовать приложение для работы с электронными таблицами, например Excel, для небольших проектов с меньшей сложностью и меньшим направлением. Поскольку электронные таблицы не имеют достаточный уровень безопасности и довольно тривиальны, когда дело доходит до операций с базой данных, в данное время используются реляционные базы данных для хранения отчетов об ошибках и представления их через интерфейс отслеживания ошибок. Система отслеживания ошибок помогает командам QA в циклах разработки программного обеспечения.

Компании по разработке программного обеспечения, безусловно, используют систему отслеживания ошибок, поскольку это необходимо. Система для отслеживания ошибок действительно требуется при разработке каждого программного продукта, поскольку многие разработчики не поддерживают адекватные документации по требованиям заказчика на протяжении всего жизненного цикла продукта. Существует много преимуществ использования системы отслеживания ошибок. Четкая связь разных отделов разработки может быть обеспечена базой данных ошибок. По сравнению с неофициальными электронными письмами, хорошо написанные отчеты по определённым стандартам, очень детально объясняют текущие ошибки и их приоритеты. Однако отслеживать все текущие ошибки

непросто. Это утомительная задача. На рынке имеются уже готовые системы отслеживания ошибок. Они являются либо открытыми, либо коммерческими. Лучшая система отслеживания ошибок среди них может использоваться для удовлетворения требований клиентов. При использовании системы отслеживания ошибок, необходимо учитывать многие факторы, такие как:

- установка приложений;
- процесс отчетности;
- метод уведомления.

Прежде чем использовать систему отслеживания ошибок, она должна быть настроена. Известное приложение для управления ошибками, которое также имеет открытый исходный код, является «Bugzilla». Во многих продуктах с открытым исходным кодом, таких как Linux, Eclipse и Mozilla, он используется. Но есть вероятность возникновения проблем с конфигурацией продуктов с открытым исходным кодом. Системы слежения за ошибками, такие как Bugzilla, занимают больше времени для настройки и не кажутся удобными для пользователя. Это связано с тем, что процесс настройки очень сложный. Пользователям сложно установить и настроить такую систему на свой сервер.

Рисунок 3- внешний вид системы отслеживания ошибок «Bugzilla»

Как было указано выше, Bugzilla сложно установить и поддерживать, хотя у него есть богатый набор функций. Его пользовательский интерфейс также не вдохновляет. Кроме того, для начинающего пользователя задача установить и использовать Bugzilla достаточно нетривиальна. Несмотря на то, что для его установки и использования имеются подробные шаги, никогда не бывает легко заставить его работать в первый раз. Это типичная проблема всех продуктов с открытым исходным кодом. Что касается процесса отчетности, отчет об ошибках должен быть очень простым и эффективным для отслеживания ошибок. Шаги в отчетах об ошибках должны быть упрощены, чтобы обеспечить гладкую передачу ошибок среди команды разработки программного обеспечения. Если веб-приложение имеет много HTML-страниц и не настроено на удобную навигацию, работать с такой системой сложно. Когда дело доходит до отслеживания ошибок в таких веб-приложениях, случаются проблемы, поскольку в системе существуют проблемы с навигацией, и люди, участвующие в отслеживании ошибок,

расстраиваются в данной системе отслеживания ошибок. Сложность таких систем может быть уменьшена путем разработки плавной и удобной навигации. В такой системе разработчикам легко работать с отслеживанием ошибок программного обеспечения. Некоторые системы отслеживания ошибок делают процесс отчетности очень утомительной задачей. Вместо того, чтобы снизить нагрузку на людей, они в действительности, увеличивают его.

Любая система отслеживания ошибок может быть двух типов: простые и сложные. Однако в идеале такая система должна быть простой и эффективной. Она должна быстро работать, а так же помогать людям отслеживать и исправлять ошибки, чтобы создавать качественные продукты и удовлетворять запросы клиентов.

«BugGenie» - это еще одна существующая система отслеживания ошибок с очень хорошими функциями. Однако, процесс отчетности ошибок в этой системе довольно сложен. Кроме того, пользователям, участвующим в разработке программного обеспечения, придется добавить так много вещей в отчет об ошибках, что сделает этот отчет намного более сложным. Еще одной важной особенностью системы отслеживания ошибок должны быть уведомления. Даже когда разработчики программного обеспечения не заносятся в систему, они должны получать уведомления о некоторых событиях и чтобы быть частью общего процесса работы системы. Уведомления помогают членам команды понять изменения в отчетах об ошибках и сроки для исправления ошибок. Уведомления являются частью автоматизированной системы отслеживания ошибок. Например, когда сообщается о новой ошибке, все заинтересованные стороны, связанные с отслеживанием ошибок, должны быть немедленно уведомлены.

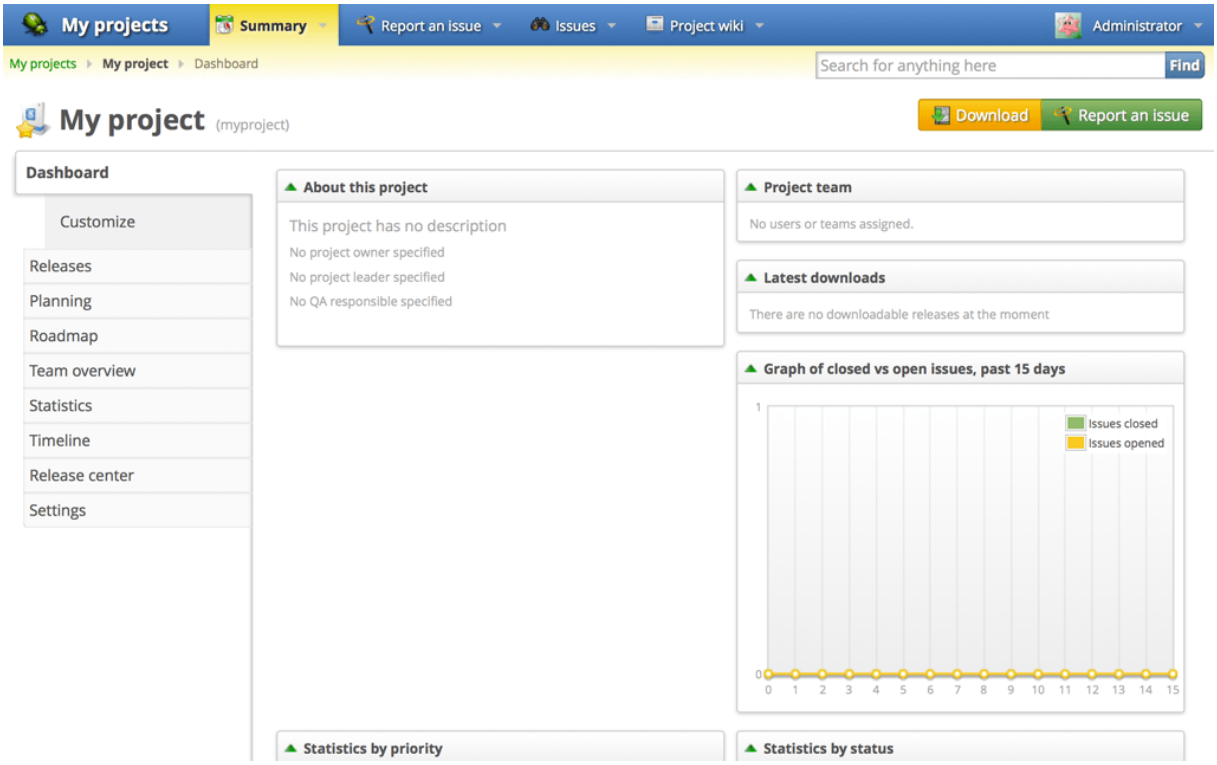

Рисунок 4- внешний вид системы отслеживания ошибок «BugGenie»

Многие системы отслеживания ошибок, такие как «BugTracker.NET» и «Flyspray», в качестве уведомлений отправляют письма. Тем не менее, фильтры спама иногда могут блокировать их, что делает уведомления по электронной почте ненадежными. RSS (ReallySimpleSyndication), предоставляемый технологией Web 2.0, - это еще один способ реализовать доставку уведомлений. Когда пользователи получают уведомления через RSS-каналы, спам-фильтры не блокируют их. Подписчики контента RSS автоматически получают обновления, и, таким образом, заинтересованные стороны процесса отслеживания ошибок осведомлены об уведомлениях. Когда RSS используется эффективно и правильно, пользователи быстро получают уведомления об изменениях, и они не обязательно должны быть в системе всегда. Именно по этой причине система отслеживания ошибок должна иметь возможность рассылать уведомления по RSS-каналам, которые максимизируют эффективность коммуникации среди участников, участвующих в отслеживании ошибок.

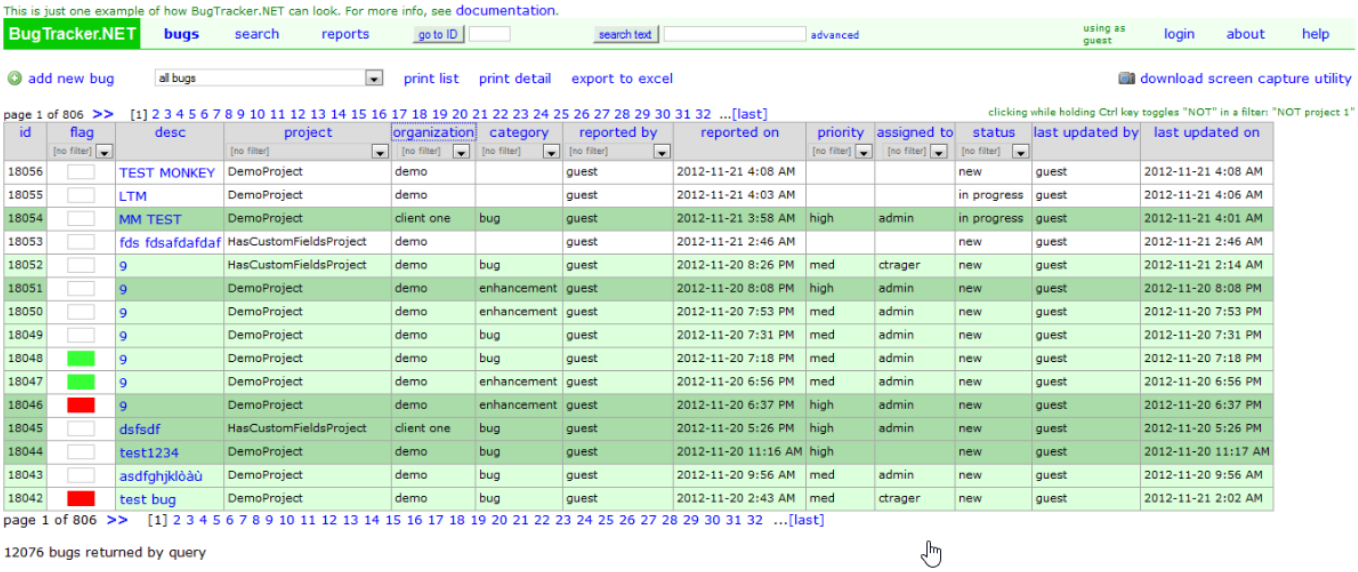

Рисунок 5 – внешний вид системы отслеживания ошибок «BugTracker.NET»

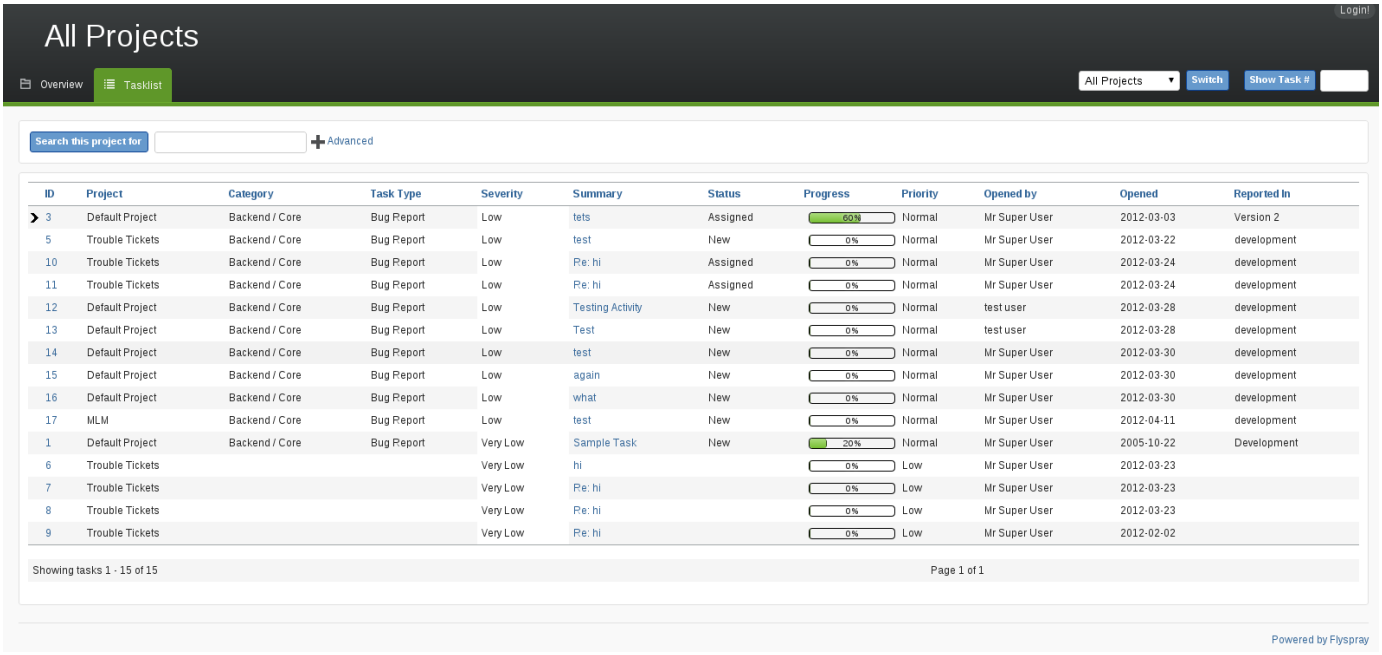

Рисунок 6 – внешний вид системы отслеживания ошибок «Flyspray»

5. Эффективные системы отслеживания ошибок

Инженеры-разработчики программного обеспечения часто участвуют в исправлении ошибок в разрабатываемой системе. Время, затрачиваемое на это, может быть уменьшено, а качество программного обеспечения может быть увеличено за счет использования эффективной системы отслеживания ошибок. Исходная информация об ошибке может помочь инженерам быстрее исправлять ошибки, тем самым экономя время разработки и стоимость. С течением времени команда разработчиков может сократить массу потерь времени в отношении отслеживания и исправления ошибок с помощью сложной системы отслеживания ошибок. Учитывая эти факты, мы можем составить следующую концепцию предлагаемых направлений совершенствования систем отслеживания ошибок.

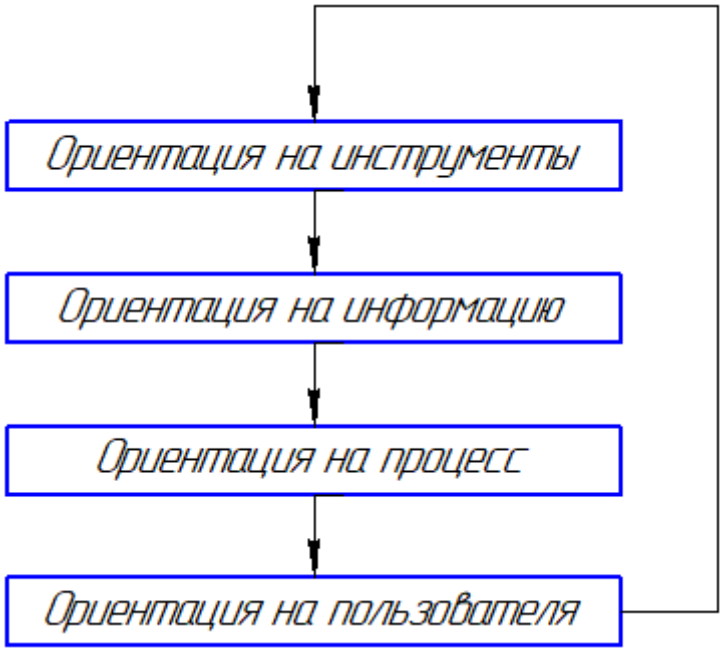

Рисунок 7 – Предлагаемые направления совершенствования систем отслеживания ошибок

Как можно видеть на рис. 7, предлагаемые направления ориентированы на инструменты, информацию, процесс и пользователя. Это итеративные

направления, которые можно циклически возвращать после каждой итерации.

5.1 Ориентация на инструменты

Ориентация на инструмент означает, что системы отслеживания ошибок могут быть настроены для неявного сбора трассировки стека и добавления его в отчет, содержащий сведения об ошибке. При этом он может улучшить возможности сбора информации. Он также может использовать шаги, чтобы использовать макрорекордеры, инструменты для захвата или воспроизведения, результат работы которых позже могут наблюдать разработчики программного обеспечения. Из этих наблюдений можно зафиксировать тестовые примеры и легко исправить ошибки. Инструментальная функция может улучшить возможности системы отслеживания ошибок с точки зрения сбора информации, которая является весьма актуальной и может быть легко использована для исправления ошибок в системе.

5.2 Ориентация на информацию

Это еще одно направление, которое должно помочь разработчикам программного обеспечения лучше сосредоточиться на сборе информации, которая должна храниться в отчетах об ошибках. С этой целью некоторые инструменты, такие как Cuezilla, могут быть встроены в системы отслеживания ошибок. Такие инструменты проверяют информацию, содержащуюся в отчетах об ошибках, и обеспечивают обратную связь, которая помогает улучшить качество информации.

5.3 Ориентация на процесс

Это направление предназначено для усовершенствований, ориентированных на процесс. Процесс, выполняемый в системах отслеживания ошибок, может быть дополнительно улучшен с использованием этого направления. Это означает, что все административные

действия, связанные с отслеживанием ошибок и их исправлением, относятся к процессинговой функции. Системы отслеживания ошибок, ориентированные на процесс, также могут сосредоточиться на разработчике, которому поручено исправлять ошибки, помимо составления исчерпывающего отчета об ошибках. Другие преимущества ориентации на процесс - это то, что разработчики могут лучше понимать сообщения об ошибках и, таким образом, они знают о возможных действиях, которые необходимо предпринять для их решения; это также помогает разработчикам оценивать время, затрачиваемое на конкретные ошибки, и соответственно планировать свое время. Функция, ориентированная на процесс, может влиять на пользовательскую функцию.

5.4 Ориентация на пользователя

Под пользователем подразумеваются разработчики программного обеспечения, а так же QA команда. В этом направлении основное внимание уделяется обучению QA команды, чтобы они могли собирать необходимую информацию, а так же понимали как это сделать. Эти действия помогают как разработчикам программного обеспечения, так и QA команде. Ожидаемая информация в отчете об ошибке заставит разработчиков быстрее понять и быстро исправить ошибки в программном обеспечении.

Вышеупомянутые направления обеспечивают систематический подход в улучшении систем отслеживания ошибок.

5.5 Вопросы в отчете об ошибке

Когда любой инженер-программист представляет отчет об ошибке, скорее всего, ему задают много вопросов. Типичные вопросы, которые задаются в таких случаях:

- название продукта;
- что за ошибка;

- в каком компоненте возникла ошибка;
- в каком методе возникла ошибка;
- на какой платформе работает приложение;
- на какой ОС установлено приложение.

Информация, предоставленная разработчиком, сообщившим об ошибке, может быть неполной изначально. Когда отчет об ошибках отправляется разработчиком, уточняющие вопросы должны быть заданы сразу же. Командам разработчиков программного обеспечения рекомендуется иметь системы отслеживания ошибок, содержащие «сборку экспертных систем». Эти системы задают все необходимые вопросы инженерам-программистам, чтобы автоматизировать работу. Вопрос, который должен быть задан разработчику, не является статичным. Вопросы не возникают последовательно. Более того, ответ на вопрос определяет следующий возможный вопрос. Для создания экспертной системы необходимы следующие данные:

- Информация о местоположении ошибок, которая имеет решающее значение для их решения. Местоположение дает вам номер строки, метод, класс и т. Д. Это помогает разработчикам легко перемещаться в нужное место. Многие среды разработки программного обеспечения (IDE) позволяют обнаруживать ошибки одним щелчком кнопки или щелчком.

- Из списка ошибок можно построить модели машинного обучения, которые выбирают вопросы, а также предсказывают местоположение ошибки на основе данных ответов. Таким образом, мы получаем сбор информации, которая необходима при внедрении инструмента, который может поддерживать автоматическую оценку информации.

Как происходит отлаживание программного обеспечения микроконтроллера? Обычно, используются JointTestActionGroup и осциллограф. Чаще всего так и происходит. Но стоит учитывать, что микроконтроллеры неимоверно разнообразны в наше время. Они выполняют множество различных задач. Сейчас мы рассмотрим, какие существуют особенности тестирования сложного программного обеспечения для микроконтроллеров.

6. Особенности тестирования управляющих систем

Какие же существуют незаурядные особенности тестирования микроконтроллеров? Например, если микроконтроллер работает с силовым оборудованием, такой микроконтроллер нельзя остановить. Так как он управляет силовыми ключами при помощи широтно-импульсной модуляции, а так же занимается регулировкой огромного количества параметров, таких как напряжение звена постоянного тока, положение рабочего органа, частота вращения и т.д. При остановке микроконтроллера во время его работы, последствия будут катастрофичными. Начиная от простого отключения микроконтроллера и заканчивая буквальным взрывом. В данном случае, при тестировании работы такого микроконтроллера можно будет только проверить только часть программного обеспечения перед реальным запуском всех систем. Так же значительное влияние оказывает сложность низкоуровневого программного обеспечения.
Величина одной секции кода программного обеспечения в 40-200 Кбайт это распространённая ситуация для промышленного микроконтроллера. Такой микроконтроллер невозможно проверить банально подключив к нему осциллограф. Не стоит забывать про такую проблему, физическая недоступность микроконтроллера. Пояснение: зачастую микроконтроллер устанавливается где-то внутри силовой установки, корпус которой

герметично запаян. Поэтому, до таких микроконтроллеров физически затруднительно добраться. Серьёзным препятствием является так же полное отсутствие какой-либо операционной системы. Поскольку, микроконтроллеры, управляющие силовым оборудованием, работают в реальном времени, подходящих операционных систем для работы с ними не существует. И финальная особенность – системы управления силовыми установками самого низкого уровня реализованы чаще всего системами автоматической регулировки. В частности, программное обеспечение составлено из многочисленных замкнутых структур с пропорционально-интегрально-дифференцирующим регуляторами, мертвой зоны, блоками насыщения, и т.д. Вычисления в таком программном обеспечении выполняются с некоторой заданной частотой. Например, при частоте 15кГц срабатывает прерывание, и все перечисленные блоки начинают пересчитываться. После пересчета, эти блоки образуют необходимую структуру для управления устройством. При такой работе алгоритмов управления провести пошаговую отладку не представляется возможным, так как каждый модуль по отдельности показывает тот результат, которого от него не ожидают. В итоге, для исчерпывающей проверки необходима настройка сотен различных параметров.

7. Как же происходит процесс тестирования?

Вариант с осциллографом отметается сразу же. Безусловно, он может подойти для тестирования неких аппаратных проблем, но при этом сотки внутренних переменных внутри системы останутся в тени. Поэтому, даже если с помощью осциллографа будут найдены какие-либо неполадки, понять причину их возникновения не получится.

Второй самый очевидный вариант – составить модель тестируемого устройства. Зачастую, именно с разработки модели начинается проектирование сложных управляющих структур. Используя специальное

программное обеспечение, например SimulinkMatlab, можно составить точную модель нужного устройства. Стоит отметить и возможность полного описания тестируемого устройства посредством языка программирования. Удобство этого способа заключается в том, такой код программного обеспечения после отладки можно сразу же загрузить в микроконтроллер. Но проблема в том, что зачастую такие модели отличаются от реальных устройств, из-за чего программное обеспечение, до этого идеально работавшее в модели, в реальности показывает совершенно не тот результат, что мы ожидаем от него. Частый случай – объект тестирования слишком сложен, поэтому писать программное обеспечение для него приходится прямо на объекте.

Существуют программы, которые позволяют использовать готовые блоки программного кода, которые поддерживаются топовыми решениями от знаменитых производителей. В теории, можно составить программное обеспечение для нужного нам микроконтроллера, а потом просто загрузить его. На практике же, такой метод почти не используется. Главная причина отказа – это малый контроль над тем, что происходит внутри микроконтроллера под управлением такого программного обеспечения. Таким образом, если программное обеспечение не будет работать, или будет работать не так, как он него ожидается, никто не сможет разобраться, в чем причина и устранить недостатки программного кода. Так же, стоить отметить, что при таком подходе, никто не заботится о производительности такого программного кода.

После всего вышесказанного мы делаем такой вывод: программное обеспечение для сложных микроконтроллеров необходимо писать вручную, а тестировать непосредственно на объекте. Само тестирование реализуется только проверкой всех внутренних переменных осциллографом. То есть выводом временных графиков, где будет видно, как каждая переменная меняется с течением времени. При этом, не рекомендуется ставить любой

внешний интерфейс, так как все ресурсы нашего микроконтроллера уйдут на обслуживание этого внешнего интерфейса. Безусловно, невозможно зафиксировать сращу все переменные. Но нескольких массивов вполне должно хватить. В первом опыте мы рассматриваем первые несколько переменных, затем они заменяются на следующие. Данный процесс повторяется до тех пор, пока программное обеспечение не будет просмотрено полностью, или пока не будет обнаружена ошибка в программном коде.

Возникает справедливый вопрос, как мы можем снять подобные осциллограммы? Например, такая компания как TexasInstruments разработали свою среду разработки и позволили производить тесты своих микроконтроллеров в режиме реального времени, без остановки самого микроконтроллера. При этом, эта среда разработки сама может строить нужные осциллограммы, без подключения дополнительных внешних интерфейсов. Но и у такого способа тестирования есть накладки. Как пример, такие осциллограммы показываются в своих текущих значениях, без какого-либо масштабирования и в разных окнах.

8. Исследование тестирования программного обеспечения: достижения, проблемы, цели

Разработка программного обеспечения охватывает несколько дисциплин, предназначенных для предотвращения и устранения неисправностей, а также для обеспечения адекватного поведения. Тестирование, предмет этой главы, является широко распространенным подходом к оценке в промышленности, но он по-прежнему в значительной степени является дорогостоящим, а его эффективность непредсказуема. Действительно, тестирование программного обеспечения - это широкий термин, охватывающий различные виды деятельности в рамках цикла разработки и за его пределами, направленные на достижение различных целей. Следовательно, исследования тестирования программного обеспечения сталкиваются с множеством проблем. В этой главе представляется блок-схема самых актуальных проблем, которые необходимо решить.

Суть тестирования

Тестирование - это важная деятельность в области разработки программного обеспечения. Проще говоря, это означает наблюдение за работой программного обеспечения с целью проверки, ведет ли оно себя по как ожидается, или имеются потенциальные сбои. Тестирование широко используется в промышленности для обеспечения качества: действительно, путем непосредственного изучения программного обеспечения при его работе, мы получаем реалистичную обратную связь о поведении.

Тем не менее, помимо очевидной прямоты проверки выборки тест ранов, тестирование охватывает множество видов деятельности, техники и участников и создает множество сложных задач. В самом деле, со всей сложностью, постоянством и критичностью постоянно растущего

программного обеспечения, обеспечение того, чтобы он вел себя в соответствии с желаемыми уровнями качества и надежности, становится более важным и все более сложным и дорогостоящим. Ранние исследования подсчитали, что тестирование может потреблять 50% или даже больше затрат на разработку, а недавний подробный опрос в Соединенных Штатах дает количественную оценку высоких экономических последствий неадекватной инфраструктуры тестирования программного обеспечения. Соответственно, возникают новые исследовательские проблемы, например, как примирить вывод на основе моделей тестовых примеров с современными динамически развивающимися системами или как эффективно выбирать и использовать данные времени выполнения тестов. Эти новые возникающие проблемы направлены на увеличение давних проблем, таких как, как квалификация и оценка эффективности критериев тестирования, или как минимизировать количество повторных тест ранов после изменения программного обеспечения.

В последние годы эта тема привлекла все больший интерес со стороны исследователей, о чем свидетельствуют многочисленные специализированные мероприятия и семинары, а также растущий процент тестовых документов на конференциях по разработке программного обеспечения. Например, на 28-й Международной конференции по разработке программного обеспечения (ICSE 2006) четыре из двенадцати конференций в исследовательском треке, были посвящены «Тестированию и анализу».

Эта глава организует множество выдающихся исследовательских задач для тестирования программного обеспечения в последовательную блок-схему. Определенные блоки представляют собой набор из четырех конечных и недостижимых целей.

Различные аспекты тестирования программного обеспечения

Тестирование программного обеспечения - это широкий термин, охватывающий огромный спектр различных видов деятельности: от тестирования небольшого фрагмента кода разработчиком (модульное тестирование) до проверки клиентами большой информационной системы (приемочные испытания). На разных этапах могут быть разработаны тест-кейсы с целью достижения различных целей, таких как:

- выявление отклонений от требований пользователя;
- оценка соответствия стандартной спецификации;
- оценка устойчивости к стрессовой нагрузке;
- выявление уязвимостей безопасности;
- измерение данных атрибутов, таких как производительность;
- удобство использования;
- оценка эксплуатационной надежности.

Вследствие этого множества целей и масштабов возникает большое количество значений для термина «тестирование программного обеспечения», что порождает множество специфических задач исследования. Рассматривая типы тестирования, и ставя правильные вопросы, мы можем конкретизировать различные случаи, различая конкретные аспекты, которые могут характеризовать тот или иной тип тестирования.

ПОЧЕМУ: почему мы делаем наблюдения? Этот вопрос касается цели тестирования, например:

- мы ищем недостатки?
- нам нужно решить, может ли продукт быть выпущен?
- нам нужно оценить удобство использования пользовательского интерфейса?

КАК: какую часть кода мы наблюдаем, и как мы её выбираем? Это проблема выбора теста, которая может быть выполнена для этого случая, наугад или систематически, применяя некоторые алгоритмические или статистические методы. Это важный вопрос, так как выбор тестовых примеров и критерий теста - сильно влияет на эффективность теста.

ЧТО: что мы тестируем? Учитывая систему, которую мы тестируем, мы можем наблюдать за её работой либо в целом, либо, фокусируясь только на её части, которая может быть более или менее большой (единичный тест, тест компонента, интеграционный тест). Этот вопрос приводит нас к различным уровням тестирования.

ГДЕ: где мы выполняем тестирование. Тестирование может выполняться в различных местах, на внутренних сервер компании, или на внешних серверах заказчика. Этот вопрос предполагает наивысшую актуальность, когда речь идет о тестировании встроенных систем.

КОГДА: когда в жизненном цикле продукта мы выполняем тесты? Обычным ответом является: чем раньше, тем лучше, поскольку стоимость устранения сбоев увеличивается по мере продолжения жизненного цикла. Но некоторые тесты, в частности те, которые зависят от окружающего контекста, не всегда можно провести в «лаборатории», и мы не сможем вести какое-либо значимое наблюдение до тех пор, пока система не будет развернута и не будет работать.

Эти вопросы представляют собой очень простую и интуитивно понятную схему характеристик программных проверок, которые могут

помочь в организации блок схемы для будущих задач в области исследований.

Блок-схема разработки программного обеспечения

В блок-схеме предусмотрены направления для достижения желаемого места назначения, начиная с красной точки «вы здесь». Программа разработки программного обеспечения для тестирования организована следующим образом:

- Красная точка «вы здесь», состоит из самых заметных достижений прошлых исследований (но обратите внимание, что некоторые из этих исследований все еще продолжаются);

- Финальный пункт назначения изображается в виде набора (четырех) недостижимых целей: мы используем этот термин для обозначения асимптотических целей в конце четырех идентифицированных маршрутов. Они недостижимы по определению;

- В середине - проблемы, с которыми сталкиваются текущие и будущие исследования тестирования, на более или менее зрелой стадии, и с более или менее хорошими шансами на успех. Эти проблемы представляют собой направления, которые следует соблюдать в путешествии к финальному пункту блок-схемы, и, как таковые, они являются центральной, самой важной частью дорожной карты.

Блок-схема представлена на Рисунке 8.

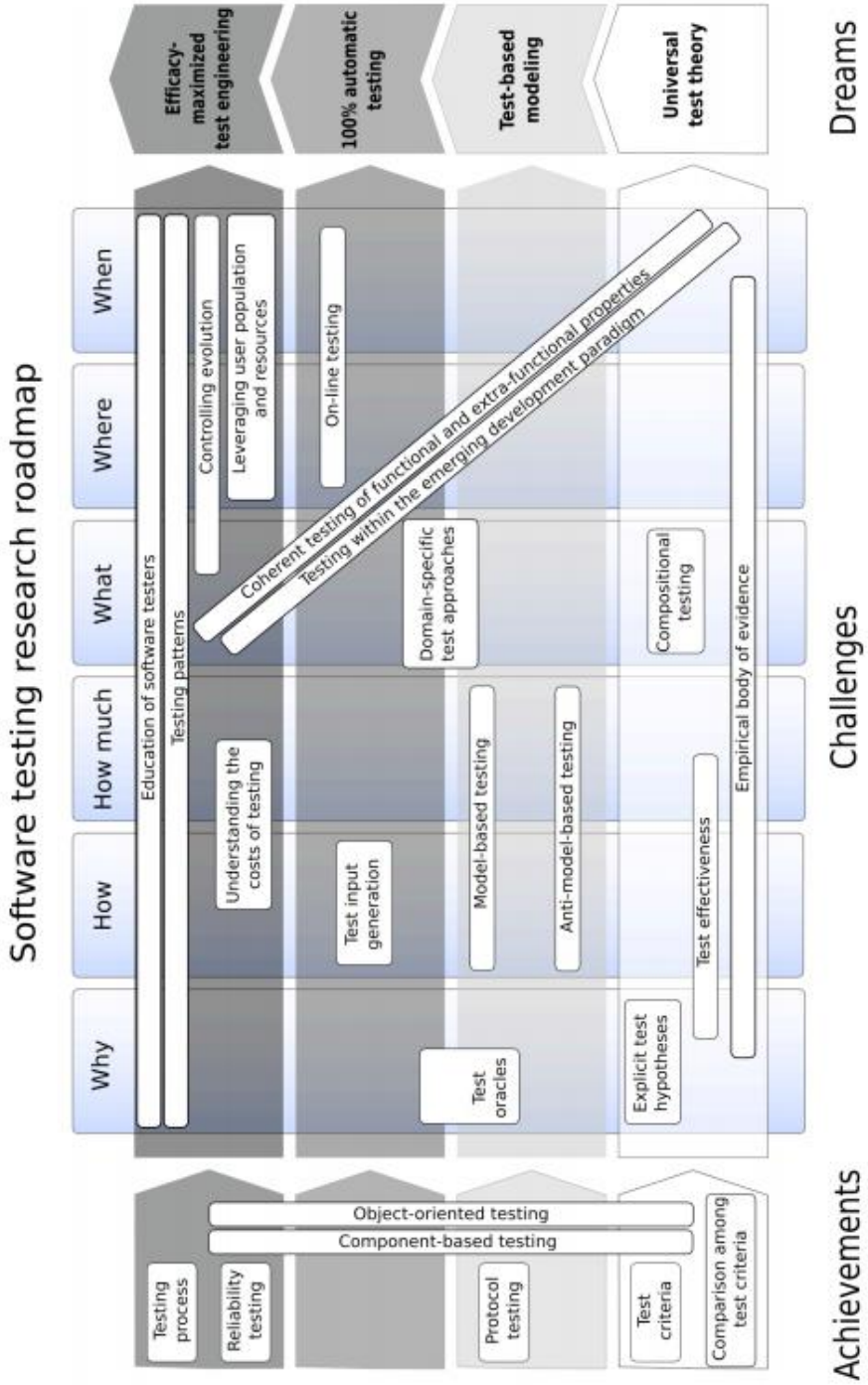

Рисунок 8 – Блок-схема путей тестирования

На этой блок-схеме мы видим новые и текущие направления исследований в центре, с исследованные темы - слева и конечные, недостижимые целям - справа. Четыре горизонтальные полосы изображают

идентифицированные маршруты исследований к финальным целям, а именно:

- Универсальная теория тестирований;
- Тестирование на основе моделирования;
- 100% автоматическое тестирование;
- Оптимизация эффективности.

Текущие достижения

Прежде чем описывать будущие маршруты исследований тестирования программного обеспечения, опишем темы, которые составляют совокупность знаний в тестировании программного. В блок-схеме они представлены с левой стороны.

Процесс тестирования

Безусловно, большинство ранних исследований эволюционировали в техники и инструменты, которые помогают спроектировать так называемый «тест-дизайн» более систематическим и включить его в процесс разработки. Для промышленного внедрения было предложено несколько моделей тестовых процессов, среди которых, вероятно, «V-модель» является самой популярной. Совсем недавно, V-образная модель поэтапного и документированного тестового процесса, по мнению некоторых, была неэффективной и излишне бюрократической, и, напротив, пропагандировались более гибкие процессы. Создание подходящего процесса тестирования было включено в FOSE2000 среди фундаментальных тем исследований, и действительно, это остается активным исследованием сегодня.

Критерии теста

Здесь описывается чрезвычайно богатый набор критериев тестирования, разработанный прошлыми исследованиями, чтобы помочь систематически идентифицировать тест-кейсы. Традиционно они различались между белым ящиком (тестирование с исследованием кода программного обеспечения) и черным ящиком (тестирование без исследования кода программного обеспечения), в зависимости от того, используется ли исходный код во время тестирования.

Сравнение критериев тестирования

Параллельно с исследованием критериев отбора тестов и достаточности теста много исследований посвящено оценке относительной эффективности различных критериев испытаний и особенно факторов, которые делают один метод лучше, чем другой.

Объектно-ориентированное тестирование

Действительно, в любой конкретный период доминирующая парадигма разработки программного обеспечения ускоряла исследования тестирования для адекватных подходов. В 90-е годы основное внимание уделялось тестированию объектно-ориентированного (ОО) программного обеспечения. Миф о том, что расширенная модульность и повторное использование, могут предотвратить необходимость тестирования был отвергнут, а исследователи вскоре поняли, что не только всё, что уже известно о тестировании программного обеспечения в целом, также применяется к ОО-коду, но и разработка ОО ввела новые риски и трудности , которые увеличивали необходимость и сложность тестирования. В частности, среди основных механизмов разработки ОО, инкапсуляция может помочь скрыть ошибки и сделать тест сложнее; наследование требует обширного повторного тестирования унаследованного кода; полиморфизм и динамическая привязка

требуют новых моделей покрытия. Кроме того, для обработки сложного спектра возможных статических и динамических зависимостей между классами необходимы соответствующие стратегии для эффективного инкрементного тестирования интеграции.

Тестирование на основе компонентов

В конце 90-х годов разработка компонентов возникла как конечный подход, который обеспечил быструю разработку программного обеспечения с меньшим количеством ресурсов. Тестирование в рамках этой парадигмы привело к новым трудностям, которые были как технические, так и теоретические. С технической стороны компоненты должны быть достаточно универсальными для развертывания в разных платформах, поэтому пользователь компонента должен повторно протестировать компонент в выбранной системе, где он развернут. Но решающей проблемой здесь является отсутствие информации для анализа и тестирования компонентов, разработанных извне. Фактически, хотя интерфейсы компонентов описаны в соответствии с конкретными моделями компонентов, они не предоставляют достаточной информации для функционального тестирования. Поэтому исследование показало, что соответствующая информация или даже сами тест-кейсы (как и при встроенном тестировании) должны быть упакованы вместе с компонентом для облегчения тестирования пользователем компонента.

Тестирование протоколов

Протоколы - это правила, которые регулируют связь между компонентами распределенной системы. Их необходимо точно определить, чтобы облегчить взаимодействие. Тестирование протокола направлено на проверку соответствия реализации протокола в отношении их спецификаций. Последние выпускаются стандартными организациями или консорциумами

компаний. В некоторых случаях также выпускается стандартный комплект тестов соответствия.

Надежность

Учитывая вездесущность программного обеспечения, его надежность, то есть вероятность безотказной работы в течение определенного периода времени в заданной среде, сегодня влияет на любой технологический продукт. Тестирование надежности подразумевает, что мы никогда не сможем обнаружить все ошибки, следовательно, используя эксплуатационный профиль для тестирования, оно пытается устранить те ошибки, которые проявляются чаще: интуитивно тестер имитирует, как пользователи будут использовать систему. Надежность программного обеспечения обычно выводится на основе моделей надежности: следует использовать разные модели в зависимости от того, удалены ли обнаруженные ошибки, и возрастает ли в данном случае надежность или нет.

Исследования в области надежности программного обеспечения во многом способствовали исследованиям в области тестирования программного обеспечения. Модели надежности программного обеспечения активно изучались в 80-х и 90-х годах. Эти модели развились и могут быть спроектированы в процессе испытаний, обеспечивая количественное руководство по тому, как и сколько нужно проверить.

К сожалению, практика тестирования надежности не продолжалась с той же интенсивностью, вероятно, потому, что она воспринимается как сложная и дорогостоящая деятельность. Однако сегодня спрос на надежность и другие качества надежности растет, и поэтому возникает необходимость в практических подходах к когерентному тестированию функционального и функционального поведения современных программно-интенсивных систем.

Маршруты

В этом разделе описываются конечные цели тестирования программного обеспечения и для каждой из них должны быть решены некоторые соответствующие задачи, направленные на то, чтобы продвинуть современное состояние ближе к самой цели.

Цель: Универсальная теория тестирования

Одной из давних целей тестирования программного обеспечения было создание прочной всеобъемлющей теории, которая полезна для резервного копирования и питания тестовой технологии. Прося «универсальную» теорию тестирования, я имею в виду одну последовательную и строгую структуру, к которой тестеры могут относиться к пониманию относительных сильных сторон и ограничений существующих методов испытаний, и руководствоваться при выборе наиболее подходящего или их сочетания, учитывая настоящие условия.

Семинарная работа по теории тестирования программного обеспечения восходит к концу 70-х годов, когда впервые были введены соответствующие понятия «надежного» или «идеального» набора тестов. Благодаря этой новаторской работе у нас есть логические аргументы, чтобы подтвердить очевидный факт, что тестирование никогда не может быть точным. Для достижения этой цели нужно решить несколько практических задач.

Задача: Явные тестовые гипотезы

В конечном счете, учитывая, что тестирование обязательно основано на аппроксимациях, эта универсальная теория также должна определять каждый метод, или тестовую гипотезу.

За исключением нескольких формальных подходов к тестированию, тестовые гипотезы обычно остаются неявными, в то время как было бы крайне важно сделать их явными. Таким образом, если мы проведем

«исчерпывающее» тестирование в соответствии с выбранным критерием тестирования, от успешного завершения тестовой кампании мы можем с полным основанием сделать вывод, что программное обеспечение является правильным в соответствии с заявленными гипотезами. То есть, мы все еще знаем, что на самом деле программное обеспечение может быть неисправным, но мы также знаем, что мы предположили истинным в начале координат и могли вместо этого быть ложными.

Такие исследования следует расширить, чтобы охватить другие критерии и подходы. Тестовые гипотезы должны быть модулированы по результатам тестирования: при тестировании на надежность, при тестировании на отладку и т. д.

Задача: эффективность тестирования

Чтобы создать полезную теорию для тестирования, нам необходимо оценить эффективность существующих и новых критериев тестирования.

Задача: композиционное тестирование

Постоянно растущая сложность программного обеспечения сильно усложняет тестирование и препятствует прогрессу в отношении любых исследовательских достижений, включая теорию тестирования. В ходе многих прошлых исследований были рассмотрены методы и инструменты, помогающие поэтапным стратегиям тестирования в организации и выполнении разных компонентов.

Задача: Эмпирическое доказательство

Сегодня важность экспериментов по продвижению дисциплины тестирования в области программного обеспечения, безусловно, не обязательно должна быть подчеркнута. В каждой теме исследований в области программного обеспечения эмпирические исследования необходимы для оценки предлагаемых методов и практик, для понимания того, как и

когда они работают, и для улучшения их. Это, очевидно, справедливо и для тестирования, когда контролируемое экспериментирование является необходимой методологией исследований.

Действительно, экспериментируя, мы должны стремиться к созданию эмпирического знания, которое лежит в основе построения и развития теории тестирования. Нам нужно изучить факторы, которые могут быть использованы для ранней оценки, где возникают неполадки, чтобы ресурсы тестирования могли быть правильно распределены. И для этого нам нужны осмысленные эксперименты с точки зрения масштаба, используемых предметов и контекста, что не всегда реалистично.

Цель: Тестовое моделирование

В настоящее время большое количество исследований сосредоточено на тестировании на основе моделей, о котором мы поговорим ниже. Ведущей идеей является использование моделей, в разработке программного обеспечения для управления процессом тестирования, в частности, для автоматического создания тест-кейсов.

Но если нам разрешено рассматривать цель, с точки зрения тестера, идеальная ситуация заключается в том, чтобы понять, что происходит сначала, а что происходит после этого. Вместо того чтобы брать модель и видеть, как мы можем лучше всего ее использовать для тестирования, давайте рассмотрим, как мы должны идеально составить модель, чтобы программное обеспечение могло быть эффективно протестировано. Было бы неплохо, если бы разработчики осознали важность и сложность тщательного тестирования на основе моделей заранее и получили соответствующие модели.

По общему признанию, это просто новый термин для старой идеи, поскольку на самом деле мы можем найти уже несколько направлений

исследований, которые более или менее явно работают над достижением этой цели. С одной стороны, понятие тестового моделирования тесно связано со старой идеей «Design-for-testability», которая в первую очередь касается разработки программного обеспечения с целью повышения управляемости и улучшения наблюдения. С другой стороны можно увидеть предыдущие подходы к тестированию на основе операторов контроля. Операторы контроля, в частности, были раннее признаны в качестве полезного инструмента для улучшения тестирования, поскольку во время своей работы они могут проверять внутреннее состояние программы.

Задача: Тестирование на основе модели

Стоимость тестирования программного обеспечения возросла до такой степени, что традиционные методы тестирования становятся нерентабельными, но, к счастью, с другой стороны, все более широкое использование моделей в разработке программного обеспечения дает перспективы устранения основного барьера для принятия модельных тестов.

Тем не менее, промышленное внедрение тестирования на основе моделей остается низким, а прогнозы о прорыве исследования неутешительны. Поэтому, помимо теоретических проблем, исследователи сегодня сосредоточены на том, как преодолеть барьеры для широкого внедрения. На данный момент существуют важные технические и связанные с процессом вопросы.

Глобальная проблема заключается в том, как мы можем комбинировать различные стили моделирования. Идея состоит в том, что модели, основанные на разных парадигмах и выраженные в любых обозначениях, могут быть легко использованы в одной интегрированной среде. Нам также нужны способы сочетания с другими подходами.

Проблемы, связанные с процессом, касаются необходимости интегрировать практику тестирования на основе моделей в текущие программные процессы. Возможно, решающими проблемами здесь являются потребности в управлении тестированием, позволяющие сделать тестовые модели максимально абстрактными, сохраняя при этом возможность генерировать исполняемые тесты на одной стороне, и отслеживание требований к тестам в течение всего процесса разработки, с другой. Наконец, нам нужны промышленные инструменты для разработки и интерактивного моделирования, которые могут помочь уменьшить неадекватное образование нынешних тестировщиков.

Частным случаем тестирования на основе модели является проверка соответствия, то есть проверка того, соответствует ли тестируемая система ее спецификации. Достигнутые в данный момент результаты впечатляют по теоретическим основаниям, но многие из предложенных методов вряд ли применимы к реальным системам.

Задача: Анти-модельное тестирование

Параллельно с тестированием на основе модели несколько усилий прилагаются к новым формам тестирования, которые лежат непосредственно на анализе выполнения программ, а не на априорной модели. Вместо того чтобы брать модель, разработать план тестирования и, следовательно, сравнить результаты тестов с моделью, эти другие подходы собирают информацию от выполнения программы, либо после активного запроса какого-либо исполнения, либо пассивно во время работы, и пытаются синтезировать из них некоторые соответствующие свойства данных или поведения. Следовательно, симметрично тестированию на основе модели, мы имеем (явно или неявно), что модель выводится на основании опыта тестирования, которое мы называем анти-модельным. Под этим термином мы

ссылаемся на все различные подходы, которые посредством тестирования, обратного проектирования модели.

Задача: Тестовые предсказания

Это задача строго связана с планированием тестирования и, в частности, с проблемой получения тестовых примеров. Ставится вопрос о том, является ли результат теста приемлемым или нет. Это соответствует так называемому «предсказанию», в идеале - методу, который обеспечивает ожидаемые результаты для каждого из этих тестовых случаев.

Хотя очевидно, что выполнение теста, для которого мы не можем отличить успех или неудачу, является бесполезным тестом, проблема тестовых предсказаний по-прежнему интересует исследователей. Но такое неудовлетворительное состояние дел, с возрастающей сложностью и критичностью программных приложений является препятствием для надежной автоматизации тестирования. Действительно, точность и эффективность предсказаний сильно влияют на стоимость / эффективность тестирования: мы не хотим, чтобы ошибки тестирования проходили незамеченными, но с другой стороны мы не хотим уведомлять о многих ложных срабатываниях, которые тратят важные ресурсы. Нам нужно найти более эффективные методы для реализации и автоматизации предсказаний из доступной информации.

Цель: 100% автоматическое тестирование

Автоматизация - один из способов сохранить анализ и тестирование качества в соответствии с растущим количеством и сложностью программного обеспечения. Исследования в области программного обеспечения делают большой упор на автоматизацию производства программного обеспечения, при этом основная масса современных средств разработки генерирует все большие и более сложный код с меньшими

усилиями. Другая сторона монеты - большая опасность того, что методы оценки качества выпускаемого программного обеспечения, в частности методов тестирования, могут отставать от таких методов разработки программного обеспечения.

Большая часть текущих исследований по тестированию направлена на повышение степени достижимой автоматизации путем разработки передовых методов генерации входных данных теста, либо, путем поиска инновационных процедур поддержки автоматизации процесса тестирования.

В последнее время были предприняты довольно многообещающие шаги для реализации этой цели для модульных тестов. К сожалению, модульное тестирование часто плохо выполняется или вообще пропускается, потому оно довольно дорогостоящее. Нам нужны подходы, чтобы сделать такое тестирование возможным в рамках процессов промышленного развития. Основным компонентом дорогостоящего тестирования устройства является огромное количество дополнительного кодирования, необходимого для имитации среды, в которой будет имитироваться работа устройства, и выполнение необходимой функциональной проверки выходов устройства.

Задача: Генерация вводных тестовых данных

Исследования в области автоматической генерации вводных тестовых данных всегда были очень активными, и в настоящее время изучается так много передовых технологий, что даже посвящение всей статьи только этой теме не даст достаточного пространства для адекватного покрытия. Что вызывает беспокойство, так это то, что до сих пор все такие усилия оказывали ограниченное влияние на промышленность, где активность генерации тестов остается в значительной степени ручной.

Что касается случайной генерации тестов, то это считалось недостаточным подходом в отношении систематических методов, которые

рассматривались, как всеобъемлющие и способные находить важные кейсы, которые, вероятно, будут упускать из виду случайные методы. Тем не менее, в предыдущих исследованиях в основном сравнивались реализации случайных тестов в сложных реализациях систематических методов. Сегодня несколько исследователей предлагают умные реализации случайного тестирования, которые, как представляется, превосходят систематическое генерирование тестов. Основная идея таких подходов заключается в том, что случайная генерация динамически улучшается, используя информацию обратной связи, собранную по результатам предыдущих тестов.

Наиболее перспективным направлением является определение эффективности способов сочетания соответствующих сильных сторон систематического (основанного на модели) и случайного тестирования. Наконец, что касается генерации тестов на основе поиска, его суть состоит в том, чтобы исследовать пространство решений для выбранного критерия теста, используя метаэвристические методы, которые направляют поиск в потенциально наиболее перспективные области входного пространства. Привлекательная особенность заключается в том, что этот подход можно применить к неограниченному кругу проблем.

Задача: Онлайн тестирование

Параллельно с традиционным представлением о тестировании как деятельности, проводимой до релиза продукта, чтобы проверить, будет ли программа вести себя должным образом, появляется новая концепция тестирования вокруг идеи мониторинга поведения системы в реальной жизни, используя динамический анализ.

Фактически мониторинг выполнения в реальном времени используется уже более 30 лет, но новый интерес возникает из-за возрастающей сложности и повсеместного характера программных систем. Терминология неравномерна, и в литературе используются различные термины, такие как

мониторинг, тестирование времени выполнения, онлайн-тестирование. Все подходы направлены на то, чтобы наблюдать за поведением программного обеспечения на местах, с целью определить, соответствует ли оно его предполагаемому поведению, обнаруживает сбои или проблемы с производительностью. В некоторых случаях предпринимается попытка восстановления в режиме онлайн, в других случаях анализ проводится автономно, чтобы создать профиль или получить показатели надежности.

Одна отличительная особенность онлайнового тестирования заключается в том, что нам не нужно разрабатывать набор тестов, чтобы симулировать тестируемую систему, поскольку мы ограничиваемся пассивным наблюдением за тем, что происходит.

В середине между классическим тестированием перед установкой программного обеспечения и пассивным мониторингом мы могли также провести проактивное тестирование, то есть симулировать приложение после развертывания. Подобная идея используется в так называемой «аудиторской» структуре, предлагающей проводить этап тестирования приема для веб-сервисов.

Другая проблема связана с возможностью проведения тестирования на местах, особенно для встроенных приложений, которые должны быть развернуты в среде с ограниченными ресурсами, где накладные расходы, требуемые при тестировании аппаратуры, не могут быть осуществлены. Интересное новое направление исследований было предпринято Kapfhammer, которые разрабатывают инструмент Juggernaut для тестирования приложений Java в ограниченной среде. Их первоначальная идея состоит в том, чтобы использовать информацию, которая до сих пор использовалась для настройки набора тестов, а также для адаптации тестовой среды. Такая идея привлекательна и, безусловно, может найти множество других полезных применений.

Цель: Техническая инженерия с максимальной эффективностью

Конечной целью исследования тестирования программного обеспечения, сегодня остается то, что позволит нам экономически эффективно разрабатывать «практические методы тестирования, инструменты и процессы для разработки высококачественного программного обеспечения».

Все теоретические, технические и организационные вопросы, рассмотренные до настоящего времени, должны быть согласованы с жизнеспособным процессом тестирования, обеспечивающим максимальную эффективность. Кроме того, неотъемлемые технические возможности и сложность предлагаемых исследователями решений, должны быть скрыты за простой в использовании интегрированной средой. Эта цель ставит такие сложные задачи, что мы квалифицируем её как самую высшую цель исследования тестирования программного обеспечения.

Основным препятствием для этой достижения этой цели, является растущая сложность современных систем. Этот рост сложности влияет не только на саму систему, но и на среды, в которых эти системы развернуты.

Стратегии согласования процесса разработки, для максимизации эффективности тестирования, относятся к дизайну «тестируемости». Тестируемость - это более широкая концепция, чем просто моделирование системы, она также включает в себя характеристики реализации, а также сами тесты и их среды поддержки. Действительно, дизайн тестируемости был объяснен практиками как основной источник затрат при тестировании.

Техническая инженерия с максимальной эффективностью требует выполнения множества задач, некоторые из которых приведены ниже.

Задача: Управление развитием

Большинство проверок, проводимых в промышленности, включают повторный тест уже протестированного кода, чтобы убедиться, что изменения в программе или в контексте не оказали негативного влияния на работоспособность системы. Как указано в FOSE2000, из-за высокой стоимости регрессионного тестирования нам нужны эффективные методы для уменьшения количества повторных проверок, определения приоритетов регрессионных тестов и автоматизации повторного выполнения тестовых случаев.

В общем, нам нужны стратегии для проведения регрессионного тестирования в больших составных системах. Нам также необходимы практические подходы к регрессионному тестированию глобальных системных свойств, поскольку некоторые части системы могут быть модифицированы. Например, учитывая компонент, который изначально был разработан для одной архитектуры, нам нужно понять, как проверить, работает этот компонент при подключении к другой архитектуре. Такая проблема также имеет решающее значение для тестирования семейств продуктов.

Связанная с этим идея - это тест-факторинг, который заключается в преобразовании одного глобального системного теста в набор многих небольших модульных тестов. Эти модульные тесты могут выполнять небольшую часть системы точно так же, как это было при тестировании системы, но, будучи более сфокусированными, они могут выявлять сбои в определенных выбранных частях системы. Тест-факторинг сегодня активно исследуется, так как он обещает корректировку порядка изменений в выполнении регрессионных тестов.

Общая деталь для многих существующих подходов к регрессионному тестированию состоит в том, что для сравнения можно используется,

например спецификация требований или архитектура программного обеспечения. Для современных программных приложений, которые постоянно динамические развиваются, мы не можем заранее знать, что такие спецификация, архитектура и даже сам исходный код не поменяются. Парадигма тестирования может переходить от регрессионного тестирования к одному из непрерывных тестов.

Следовательно, важнейшая проблема заключается в том, как поддерживать контроль над качеством программного обеспечения, которое динамично развивается. Нам нужно понять, что означает регрессионное тестирование в таком эволюционирующем контексте, и как мы можем модифицировать и расширять основную идею выборочного тестирования регрессии.

Задача: Использование пользователей для тестирования

Мы уже упоминали о появляющейся тенденции непрерывной проверки после развертывания с помощью онлайновых методов тестирования, когда традиционные методы автономного тестирования становятся неэффективными. Поскольку системы с интенсивным использованием программного обеспечения могут вести себя по-разному в разных средах и конфигурациях, нам нужны практические способы расширения онлайн-тестирования для охвата широкого спектра возможных поведений. Один из перспективных подходов к решению этой проблемы заключается в увеличении собственной деятельности по обеспечению качества путем использования динамически собираемых данных. Перспектива заключается в том, что используя данные от пользователей, мы можем выявить реальные проблемы, на которых необходимо сосредоточить деятельность по тестированию. Например, предоставляя каждому пользователю разную конфигурацию по умолчанию, пользовательская база может быть использована для более быстрого выявления конфликтов или проблем

конфигурации. Несмотря на то, что начинают появляться некоторые коммерческие инициативы, такие как Microsoft’s Customer Experience Improvement Program, это направление все еще находятся в зачаточном состоянии. Одна важная исследовательская задача остается открытой - как определить эффективные методы для раскрытия потенциала деятельности большого количества пользователей, работающих на аналогичных приложениях, на взаимосвязанных машинах. Эта проблема высокого уровня включает в себя еще несколько конкретных задач, среди которых:

- Как мы можем собирать данные времени выполнения из программ, работающих в «полевых» условиях, не накладывая слишком больших расходов?

- Как мы можем хранить и добывать собранные (потенциально огромные) объемы необработанных данных, чтобы эффективно извлекать соответствующую информацию?

- Как мы можем эффективно использовать собранные данные для расширения и улучшения внутренних тестов и обслуживания?

Задача: тестовые шаблоны

Мы уже упоминали в цели о теории тестирования, что нам нужно понять относительную эффективность методов тестирования в типах ошибок, на которые они нацелены. Чтобы разработать процесс тестирования, нам необходимо собрать всевозможную информацию, чтобы иметь возможность найти наиболее эффективный шаблон для тестирования системы. Обычно это делается, когда, например, функциональное тестирование на основе требований сочетается с мерами адекватности покрытия кода. Еще одна рекомендация заключается в объединении оперативного тестирования с проверкой конкретных случаев.

Шаблоны предлагают хорошо зарекомендовавшие себя решения для повторяющихся проблем, или, другими словами, они документируют опыт решения проблем. Поскольку тестирование признано дорогим и подверженным нагрузкам, очень важно иметь такую документацию.

Задача: Понимание стоимости тестирования

Поскольку тестирование происходит не абстрактно, а в конкретном мире, с его рисками и безопасностью, а также с экономическими ограничениями, в конечном итоге мы хотим увязать процесс тестирования и методы с их стоимостью.

Каждая исследовательская статья по тестированию программного обеспечения начинается с утверждения, что тестирование - очень дорогостоящее мероприятие, но нам не хватает современных и надежных ссылок. Многие до сих пор ссылаются на количественную оценку высокой стоимости тестирования в учебниках, написанных более двадцати лет тому назад. В свою очередь, мы должны иметь возможность количественно определять прямые и косвенные затраты на методы тестирования программного обеспечения в настоящем.

К сожалению, большинство исследований в области тестирования программного обеспечения занимает нейтральную позицию, как если бы каждая найденная неисправность была одинаково важна или имела одинаковую стоимость. Это, конечно, не так. Нам нужны способы включения экономической ценности в процесс тестирования, чтобы помочь менеджерам по тестированию выбрать наиболее подходящие подходы. Исследователи представили парадигму разработки программного обеспечения на основе стоимости (VBSE), в которой запрашиваются количественные рамки для поддержки решений менеджеров программного обеспечения, которые повышают ценность поставляемых программных систем. В частности, были изучены различные аспекты обеспечения качества программного

обеспечения, в том числе основанные на стоимости и на оценке риска тестирования. VBSE касается главным образом управления процессами. Нам также необходимо будет учитывать функции оценки соотношения затрат и эффективности доступных методов испытаний. Ключевой вопрос: с учетом фиксированного бюджета тестирования, как его следует использовать наиболее эффективно?

Задача: Обучение инженеров по качеству программного обеспечения

Наконец, для тестирования программного обеспечения, как и для любой другой деятельности по разработке программного обеспечения, важнейшим ресурсом остается человеческий фактор. Помимо наличия передовых методов и инструментов и эффективных процессов, умение, приверженность и мотивация инженеров могут иметь большое значение между успешным тестовым процессом и неуспешным. Исследования должны стремиться к созданию эффективных решений, которые легко интегрируются в разработку и не требуют глубоких технических знаний. Но мы также должны работать параллельно для расширения возможностей человека. Это достигается как образованием, так и мотивацией. Инженеры должны обучаться, чтобы понять основные понятия тестирования, его ограничения и возможности, предлагаемые доступными методами. Образование должно продолжаться, чтобы идти в ногу с достижениями в области технологий тестирования.

Перекрестные задачи

Под перекрестными задачами мы определяем некоторые исследовательские тенденции, которые проходят через все четыре намеченные цели. В частности, мы обсуждаем здесь две трансверсальные задачи.

Задача: Тестирование в рамках возникающей парадигмы развития

История исследований в области программного обеспечения поэтапно связана с последующим появлением новых парадигм развития, которые обещают выпустить более качественное и менее дорогостоящее программное обеспечение. Сегодня мода - это сервис-ориентированная вычислительная техника, и для тестирования сервисно-ориентированных приложений возникает множество интересных проблем.

Разработчик сервиса, реализующий сервис, поставщик услуг, развертывающий и делающий его доступным, и интегратор услуг, создающий услуги, которые могут быть доступны другим, получают доступ к различным видам информации и имеют разные требования к тестированию. За исключением разработчика услуг, необходимо применять методы тестирования «черного ящика», поскольку детали обслуживания и реализации услуг зачастую недоступны.

На пути к обеспечению совместимости первая проблема заключается в обеспечении того, чтобы услуги соответствовали установленным стандартизованным протоколам обмена сообщениями. Например, руководящие принципы были опубликованы организацией WS-I (WebServicesInteroperability), нужны также инструменты тестирования для мониторинга и проверки того, что обмен сообщениями соответствует ожиданиям.

Онлайн тестирование, имеет особое значение для тестирования сервисов, поскольку мониторинг выполнения тестов - единственный способ наблюдать за поведением приложения. Сервис-ориентированное приложение обычно является результатом интеграции нескольких сервисов, контролируемых и принадлежащих различным организациям. В результате управление приложением распределяется, а службы, составляющие его, обнаруживают друг друга во время выполнения и могут меняться без

предварительного уведомления, поэтому никто не может предсказать получателя или поставщика данной услуги. Мониторинг услуг представляет множество тонких проблем, связанных с ухудшением производительности, производством нежелательных побочных эффектов и затрат. Нам нужно понять с одной стороны, как мы можем наблюдать за выполнением в распределенной сети без отрицательного влияния на производительность системы, с другой стороны, нам нужны средства для понимания того, что и когда нужно проверить.

Задача: когерентное функциональное тестирование и функциональные свойства

Безусловно, основная часть литературы по тестированию программного обеспечения касается тестирования функциональности, то есть проверки того, что наблюдаемое поведение соответствует логике спецификаций. Но этого недостаточно, чтобы гарантировать реальную полезность и адекватность цели тестируемого программного обеспечения: хорошо продуманное программное обеспечение должно выполнять нефункциональные задачи в зависимости от конкретной области приложения. Примечательно, что в то время как традиционное тестирование функциональности не предусматривает какого-либо представления о времени, многие функции могут зависеть от того, когда будут получены результаты, или от того, как долго они будут работать. Аналогичным образом, хотя тестирование функциональности не требует использования ресурсов и рабочих нагрузок, в определенных областях, таких как телекоммуникации, проблема с производительностью связана с основной категорией ошибок.

9. Поиск ошибок. Методы нахождения ошибок и причин неработоспособности электронных устройств

Нахождение ошибок в работе электронных устройств – это нетривиальная и многоуровневая задача. При быстром осмотре, можно найти только самые простые отклонения в работе устройства.

Порядок действий

Поскольку, наш порядок действий очень сильно зависит от информации, которой мы владеем в данный момент и детального внимания ко всем мелочам, довольно затруднительно составить некоторую универсальную блок-схему тестирования. Зачастую, выбор места начала действий имеет случайных характер. Безусловно, это не значит, что мы всегда должны действовать наугад, не некоторая случайность имеет место быть. Разумеется, при досконального и последовательном подходе к анализу проблемы, мы получим результат, который нас удовлетворит. В первую очередь, мы должны локализировать место неисправности. Вполне возможно, что первые попытки найти это место будут провальными, но нужно осознавать, наши ожидаемые результаты от тех или иных замеров. Первым делом, стоит использовать экспресс методы, например «внешний осмотр» и «выяснение истории возникновения неисправности». Такие, казалось бы, простые методы, на самом деле вносят огромный вклад и невероятно эффективны. При этом, следует так же учитывать различия между совершенно новым устройством и устройством, которое уже успело побывать в эксплуатации. Если мы имеем дело с тем устройством, что уже было в эксплуатации, мы можем сделать предположение, что ошибки в монтаже печатной платы отсутствуют, а так же то, что все компоненты платы верного номинала и типа. Для изделия, которое включается в первый раз, такие проверки будут актуальны и необходимы, причем непосредственно

перед первым включением. Универсальный алгоритм действий составить фактически невозможно, но мы можем придумать блок-схему на основе общих черт.

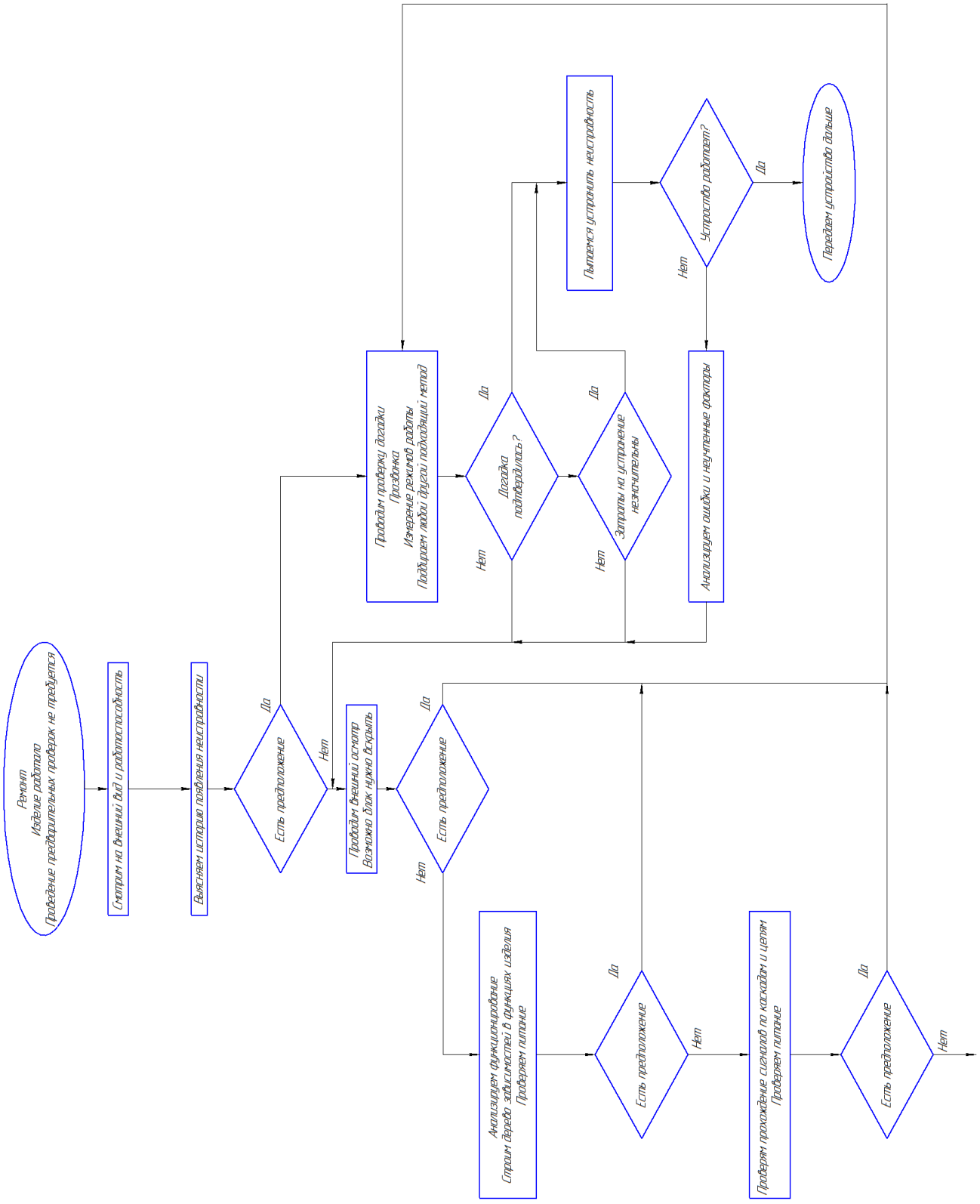

Рисунок 9 - Блок-схема на основе общих черт

Рассмотрим приведенную схему подробнее. Предварительные проверки представляют собой проверки соответствие документации, например:

- Правильный монтаж;
- Отсутствие замыканий;
- Отсутствие загрязнений.

Как мы можем понять, поиски ошибок всегда цикличны, за исключением простейших случаев. При негативном результате на очередном цикле следует применять более глубокое исследование изделия. При этом проверять работоспособность мы можем наиболее удобный для нас способом. Чаще всего это зависит от ситуации.

Применение методов

При использовании того или иного метода, мы преследуем несколько целей:

- Сбор информации;
- Проверка гипотезы о неисправности;
- Нахождение локации неисправности.

С каждым новым нашим шагом, мы получаем новую информацию, проверяем гипотезу, выведенную на основе этой информации, и пытаемся найти место неисправности. Нужно определять, что делать в первую очередь, а что во вторую. Не стоит пренебрегать информацией, полученной от вторичных действий. Если мы проверяем выдвинутую версию, то данные, полученные при этом тесте, могут подтвердить, или опровергнуть нашу догадку. Для наиболее скорого нахождения неисправностей можно использовать принцип «Раздельного теста». В первую очередь проверяем

ключевые точки схемы. Рассмотрим пример: у нас есть некоторая система, которая неисправна. В первую очередь отделяем определенный блок этой системы. Подключаем выбранный блок отдельно от основной системы. В самом блоке мы отделяем некий модуль, который так же подключается отдельно, что позволяет нам проверить уже элемент. Но у такого метода есть недостаток. Возможно, получится так, модуль исправно работает, а блок уже нет. Суть такого положения кроется в некотором несоответствии кондиций работы модуля (элемента, блока) в их системе и отдельно от неё, в лабораторных условиях. Невозможно стопроцентно воссоздать работу в системе, поэтому такие отличия будут постоянны, что бы мы не сделали. Преждевсего различия появляются здесь:

- Электрические сигналы;
- Температурная разница;
- Охлаждение.

Тем не менее, несоответствия не всегда очевидны. Обнаружение причин возникновения различных ошибок требует досконального знания тестируемого прибора, его спецификаций и принципов работы. При самых тривиальных раскладах, прибор можно разбить на структуру типа «звезда» или другую последовательную структуру. Если мы выбрали структуру типа «звезда», то нужно сначала изучить центральный узе, из полученных данных сделать вывод о его текущей работоспособности, а так же о работоспособности других узлов. Если же, по каким-либо причинам мы не можем исследовать центральный узел, то начинаем исследовать периферию. Такое исследование будет намного более трудоёмким, но таким образом мы добудем всю необходимую информацию. Если была выбрана последовательная структура, нужно убедиться, что на входе мы имеем верные сигналы, а на выходе неверные. Из чего мы сделаем вывод, что не работает структура. Далее проверка сведется к простой задаче на разделение

количества каскадов поровну и проверке сигналов в среднем каскаде. Этими действиями мы определим местонахождение предполагаемой ошибки. После чего найденную область мы опять делим пополам и повторяем ранее описанные действия.

Методы поиска и устранения неисправностей

В этой главе будут описаны практические методы для нахождения и ликвидации неисправностей в электронных устройствах. Причины неисправности могут быть следующие:

- Поломка элемента;
- Ошибка разработчиков;
- Ошибка монтажников.

Методы взаимосвязаны между собой, поэтому зачастую необходимо их одновременное применение. Так же, часто случается, что нахождение неисправности сопровождается её устранением.

Главные правила поиска неисправностей:

1. Действие не может вредить исследуемому устройству;
2. Действие приводит к ожидаемому результату;
3. Обязательно нужно отличать возможную неисправность и фактическую, предположенную причину и подтвержденную причину;
4. Обязательная правильная оценка ремонтопригодности изделия;
5. Адекватная оценка ремонта. Часто ремонт дорог, в силу чего невыгоден, но при этом в нём есть необходимость с точки зрения, изучения устройства.

Схема описания метода:

- Суть метода;
- Его возможности;
- Его достоинства;
- Его недостатки;
- Его применение.

Метод 1: Поиск информации о появления неисправности.

Суть метода: Информации о появлении неисправности очень много может рассказать о местонахождении самой неисправности и о том, на какой модуль нужно обратить внимание в первую очередь. Также данная информация позволяет сильно уменьшить затрачиваемое время на тестирование устройства и улучшить качество ремонта. Поиск информации о появления неисправности помогает понять, является ли неисправность результатом внешних факторов, например:

- Климатические условия;
- Физические/механические действия;
- Загрязнение различными веществами.

Возможности метода: Данный метод позволяет в быстром темпе выдвинуть предположение о местонахождении неисправности.

Достоинства метода: Не нужно знать какие-либо особенности самого устройства. Быстрота метода.

Недостатки метода: Необходимость получать информацию о прошедших действиях от третьих лиц. Информация так же может быть

неточной и недостоверной. Обязательна кооперация с другими методами. Есть вероятность ошибки и неточность местонахождения.

Применение метода: При условии, что неисправность в начале проявляла себя нечасто, но с течением времени интервалы её появлений стали, то большая вероятность того, что неисправен электролитический конденсатор, электронная лампа или силовой полупроводниковый элемент, чрезмерный разогрев которого приводит к ухудшению его характеристик. В случае, когда неисправность появилась после механического воздействия, ее можно попробовать определить внешним осмотром. Возможен сценарий появление неисправности после модификаций или доработок. В таком случае следует обратить пристальное внимание именно на ту часть изделия, над которой производились данные действия. О местонахождении неисправности относительно много могут сказать проявления неисправности на разных этапах ее развития.

Метод 2: Внешний осмотр

Суть метода: Внешний осмотр часто игнорируют, но именно он позволяет обнаружить местоположение примерно 50% неисправностей, особенно если это мелкосерийное производство.

Возможности метода: Метод позволяет чрезвычайно быстро обнаружить неисправность и найти ее местоположение с точностью до элемента.

Достоинства метода: Быстрота. Точное определение местоположения проблемы. Нет нужды в особом оборудовании или документации.

Недостатки метода: Можно обнаружить только те проблемы, что имеют внешнее проявление. Зачастую, требует разборки прибора. Нужен определенный опыт.

Применение метода: В первую очередь нужно смотреть на качество монтажа, а именно: правильно ли размещены элементы на плате, качество пайки, целые ли дорожки, изоляция проводов. Нужно узнать, если устройство когда-либо работало, если да, то обратить внимание на возможные следы от тепловых повреждений элементов.

Метод 3: Прозвонка

Суть метода: Проверка с помощью омметра наличия всех необходимых связей.

Возможности метода: Превентивное нахождение неисправностей во время производства. Проверка догадки о нахождении неисправности в конкретном участке.

Достоинства метода: Этот метод очень прост и не требователен к опыту. Большая надежность и точность местонахождения неисправности.

Недостатки метода: Процесс прозвонки может быть очень длительным. Обязательный доступ напрямую к элементам и контактам.

Применение метода: Проверяем наличие нужных связей и отсутствие лишних. Для комфортного использование этого метода стоит составить таблицу прозвонки, на основе электрической принципиальной схемы устройства.

Метод 4: Снятие рабочих характеристик

Суть метода: Включаем наше устройство в рабочих условиях, или имитируем такие условия, после чего сравниваем фактические характеристики с ожидаемыми. Ожидаемые характеристики можно получить из исправного устройства, спецификации, или из теоретических расчетов.

Возможности метода: Быстрая диагностика изделия целиком или частично. Примерная оценка месторасположения неисправности.

Достоинства метода: Быстрота, точность.

Недостатки метода: Наличие специального оборудования для имитации рабочих условий, или такое же исправное устройство. Высокие требования к исполнителю. Доскональное знание проверяемого устройства.

Применение метода: Как пример можно взять проверку телевизора. Мы смотрим на наличие картинки, звука. Если есть какие-то несоответствия параметров у тестируемого устройства и контрольного, мы можем судить о работе тех или иных блоков.

Метод 5: Наблюдение прохождения сигналов по каскадам

Суть метода: С применением специальной аппаратуры, например осциллографа, мы смотрим на сигналы каскадов и измеряют характеристики сигналов на контрольных точках.

Возможности метода: Возможность оценить работу изделия целиком, или по нужным нам каскадам.

Достоинства метода: Метод чрезвычайно точен и даёт исчерпывающую оценку.

Недостатки метода: Сложно провести оценку цепи, если в ней есть обратная связь. Высокие требования к исполнителю. Больше временные затраты.

Применение метода: Если в схеме есть последовательно расположенные каскады, мы смотрим, не пропадает ли верный сигнал на контрольных точках. Если есть какие-то несоответствия ожиданий и реальности мы можем судить о неисправности входа или неисправности связи.

Метод 6: Сравнение с исправным блоком

Суть метода: Берется заведомо рабочее устройство, характеристики которого потом сравниваются с неисправным устройством. Обращаем внимание на внешний вид, электрические сигналы и сопротивление. Если есть расхождения, можем выдвинуть догадку о местоположении неисправности.

Возможности метода: Быстрое диагностирование, если мы применим так же и другие методы. Можно провести ремонт не используя спецификацию.

Достоинства метода: Быстрота. Исключаются ошибки документации и моделирования.

Недостатки метода: Нужно иметь заведомо рабочее устройство. Невозможно использовать без других методов.

Применение метода: Метод крайне эффективен, так как задокументированы могут быть далеко не все характеристики прибора. Рекомендуется начать сравнение с внешнего осмотра двух устройств (расположения элементов и их конфигурации). После чего перейти к сравнению всевозможных электрических характеристик.

Метод 7: Моделирование

Суть метода: Мы моделируем работу верно работающего прибора и работу неисправного прибора. Из полученных сравнений мы выдвигаем догадку о возможных неисправностях. После чего эта догадка проверяется различными измерениями. Метод нужно применять в кооперативе с другими методами.

Возможности метода: Быстрая и относительно точная генерация догадки о местонахождении неисправности.

Достоинства метода: Точнаяоценкаработоспособности прибора. Позволяет работать с периодически пропадающими неисправностями.

Недостатки метода: Высокие требования к исполнителю. Невозможно использовать без других методов.

Применение метода: Если в устройстве наблюдается «плавающая» неисправность (неисправность, которая появляется периодически, а затем пропадает), то рекомендуется использовать именно этот метод. Для правильного моделирования нужно знать устройство тестируемого прибора, так же различные неочевидные вещи.

Метод 8: Разбиение на функциональные блоки

Суть метода: Мы разбиваем наш исследуемый прибор на составные блоки, которые проверяются отдельно друг от друга. Таким образом, мы можем быстро найти неисправный блок для дальнейшего изучения проблемы. Стоит принимать во внимание, что если исследуемый прибор сложен, то данным метод будет малоэффективен, ввиду того, что конструктивный один блок будет состоять из множества функциональных блоков.

Возможности метода: Оптимизирует работу остальных методов. Помогает определить местонахождение неисправности.

Достоинства метода: Быстрота поиска местонахождения неисправности.

Недостатки метода: Обязательно глубокое знание схемотехники исследуемого прибора. Нужно много времени для проведения анализа.

Применение метода: Этот метод применяется, если прибор состоит из блоков, которые можно быстро заменять. Меняя по очереди каждый блок на заведомо рабочий, мы сможем определить местоположение неисправности.

Метод 9: Временная модификация схемы

Суть метода: Допустимо временно менять схему прибора, например, добавлять связи/элементы, убирать связи/элементы.

Возможности метода: Точное обнаружение местоположения неисправности в цепях с обратной связью.

Достоинства метода: Позволяет уточнить местонахождение неисправности.

Недостатки метода: Обязательная модификация схемы. Доскональное знание работы тестируемого прибора.

Применение метода: Применять этот методрекомендуется**,** еслив цепи присутствуют взаимные влияния, и при этом непонятно, какое из них провоцирует возникновение неисправности. Так же метод можно использовать если есть вероятность того, что один несправный блок может повредить другие блоки. Особое внимание нужно уделять работе с обратными связями. Так как отключение или добавление обратной связи может полностью исказить работу каскадов.

Метод 10: Включение функционального блока вне системы, в условиях, моделирующих систему

Суть метода: Метод являет собой комбинацию методов «Разбиение на функциональные блоки» и «Снятие внешних рабочих характеристик». Если мы находим неисправности, блок, который мы подозреваем, проверяется вне системы. Так мы можем понять, исправен блок, или нет, и во втором случае уже обнаружить местоположение неисправности.

Возможности метода: Проверки выдвинутых предположений о работе различных частей системы.

Достоинства метода:Ремонт и тестирование возможны вне системы.

Недостатки метода: Обязательно наличие схемы для проверки.

Применение метода: Во время использования этого метода нужно наблюдать за условиями, которые мы создаем проверяя схему.

Метод 11: Предварительная проверка функциональных блоков

Суть метода: Блок тестируется за пределами системы, при использовании специального стенда. Использовать этот метод разумно, если у блока малое количество входных сигналов. Подойдут, например, блоки питания.

Возможности метода: Можно проверить работает ли блок до непосредственной установки его в систему.

Достоинства метода: Позволяет проверить основные характеристики блока без внешних воздействий. Превентивное нахождение неисправностей.

Недостатки метода: Нужна рабочая станция, чтобы осуществить тестирование.

Применение метода: Разумно применять этот метод при производстве новых устройств.

Метод 12: Метод замены

Суть метода: Блок или элемент, который мы подозреваем в неисправности, заменяться на заведомо рабочий, после чего работоспособность системы проверяется заново. Из полученных данных мы делаем выводы.

Возможности метода: Проверка наших догадок о возможных нарушениях работы элементов или блоков.

Достоинства метода: Быстрота метода.

Недостатки метода: Большое количество различных элементов или блоков требуется, чтобы применить данный метод.

Применение метода: Если работа системы после замены не меняется, значит, наша догадка была неверна. В противном случае мы смогли найти местоположение неисправности и устранить её. Возможен вариант, когда только некоторый процент от общего количества дефектов был устранен. Это означает, что мы обнаружили вторичную причину неисправности. При этом без исправления первичной причины, все наши действия будут бессмысленны, так как первичная причина выведет из строя наши замены.

Метод 13: Проверка режима работы элемента

Суть метода: Проводится сравнение токов и напряжений в заведомо правильно работающем приборе и в тестируемом. Как вариант можно сравнить токи и напряжения с их значениями в документации. После чего сделать соответствующие выводы.

Возможности метода: Нахождение местоположения неисправности с невероятно высокой точностью.

Достоинства метода: Высокая точность.

Недостатки метода: Данные измерения потребуют огромного количества времени. Высокие требования к исполнителю.

Применение метода: Этот метод применяется для измерения напряжений и токов в контрольных точках, происходит проверка падения напряжения на элементах схемы, проверяется правильная работа логических цифровых схем.

Метод 14: Провоцирующее воздействие

Суть метода: Если в тестируемом приборе имеется «плавающая» неисправность, можно попробовать оказать механическое или температурное воздействие для её выявления.

Возможности метода: Обнаружение «плавающих» неисправностей.

Достоинства метода: Тривиальность метода.

Недостатки метода: Наличие специального оборудования.

Применение метода: Начинать использование этого метода с обычного постукивания по элементам. Если нет результата, попробовать нагреть схему.

Метод 15: Проверка температуры элемента

Суть метода: Банальная проверка температуры элемента с использованием специальных приборов. Из полученных данных можно сделать предположение о неисправности элемента.

Применение метода: Особых инструкций не требуется, кроме случаев, когда мы работаем с высоковольтным оборудованием. Так как неясно, работает ли элемент при своей рабочей температуре или нет, без соответствующей спецификации или заведомо рабочим прибором.

Метод 16: Выполнение тестовых программ

Суть метода: К работающему прибору мы подключаем специальную тестовую программу, которая имеет взаимодействие со всеми компонентами системы и затем собирает информацию об их работе. Таким образом у нас будет исчерпывающие данные о работе периферийных устройств системы.

Достоинства метода: Быстрая проверка работоспособности периферии прибора.

Недостатки метода: Недостатков у данного метода довольно много. Для начала, нужно быть уверенным, что сама тестовая программа работает исправно. После чего убедиться, что ядро системы так же исправно, иначе собранные данные будут неверны.

Применение метода: Данный метод мы применяем только для финального теста и нахождения мелких недоработок.

Метод 17: Пошаговое исполнение команд

Суть метода: Подключая специализированное оборудование, микроконтроллер устройства переводят в режим пошаговой работы с машинными кодами. На каждом такте сверяются данные с шин с исправной системой или со спецификацией. Метод очень похож на «Выполнение тестовых программ», но используется в системах с любой работоспособностью.

Достоинства метода: Действует в системах с любой работоспособностью.

Недостатки метода: Высокие требования к исполнителю. Огромное количество времени на исполнение.

Применение метода: Наивысшая эффективность метода достигается на этапе разработки прибора.

Метод 18: Тестовые сигнатуры

Суть метода:

При помощи специального оборудования определяют состояние шин микропроцессорного устройства в штатном режиме работы на каждом шаге программы (или тестовой программы). Можно сказать, что это вариант пошагового выполнения программ, только более быстрый (за счет применения специального оборудования).

Достоинства метода: Возможна отладка почти неработающей системы

Недостатки метода: Большая трудоемкость. Высокая квалификация исполнителя.

Применение метода: Метод очень эффективен для отладки микропроцессорных систем на стадии разработки.

Метод 19: «Выход на вход».

Суть метода: Если у тестируемого прибора есть вход и выход, и она умеет работать в дуплексном режиме, то мы можем проверить работоспособность устройства, подавая сигнал с выхода на вход. Изучая полученные данные, такие как наличие или отсутствие сигнала и его качество, мы оцениваем пригодность соответствующих цепей.

Достоинства метода: Невероятно быстрая скорость проверки работоспособности. Небольшое количество дополнительных устройств.

Недостатки метода: Применить этот метод можно не всегда.

Применение метода: Применяем метод в самом конце тестов управляющих систем.

Метод 20: Типовые неисправности

Суть метода: На основе всего опыта ремонта конкретных приборов можно составить список типовых неполадок. У массовых приборов зачастую имеются одни и те же шаблоны поломок, изучив которые, мы можем быстро и точно найти расположение неисправности и ликвидировать её.

Достоинства метода: Метод очень быстрый, и не требует от исполнителя специальных навыков.

Недостатки метода: Необходимо сначала составить список таких неисправностей. Требуется кооперация с другими методами для подтверждения догадки.

Применение метода: Метод строится в первую очередь наличном опытеконкретного исполнителя. Случаи, когда типовые неисправности добавляются в спецификации, очень редки.

Метод 21: Анализ влияния неисправности

Суть метода: Анализируя текущую информацию о неполадках прибора, мы строим блок-схему, которая отображает работу всех блоков устройства. Изучая эту блок-схему, мы можем выдвинуть догадку о том, работа какого блока вызвала неполадку устройства. Если выдвинуть догадку мы не можем, то продолжаем собирать информацию.

Достоинства: Всю получаемую информацию нужно анализировать и систематизировать, поэтому этот метод обязателен практически для любых исследований.

Применение метода: Самый тривиальный пример – прибор не включается вовсе. Нет никакой реакции, ни запаха, ни дыма, ни звуков. Делая догадку, мы в первую очередь проверим предохранитель, как самую очевидную и простую причину. Если же он исправен, мы продолжаем искать данные.

Метод 22: Периферийное сканирование

Суть метода: Определяем контрольные точки и измеряем между ними сопротивление. Существенное отличие от метода прозвонки состоит в том, что нас интересует не только наличие или отсутствие связи, но само значение сопротивления.

Достоинства метода: Возможна внутрисхемная проверка элементов.

Недостатки метода: Нужно знать значения сопротивлений в блоке, для чего придется искать спецификацию или делать замеры на заведомо рабочем устройстве.

Применение метода: Во время использования метода нужно следить за оборудованием, которое мы применяем. Потому что оно может вывести из строя тестируемый прибор во время измерений.

10. Использование методов тестирования

Используя разработанные методики тестирования электронных устройств, были составлены тест-кейсы для управляемого трехфазного тиристорного выпрямителя.

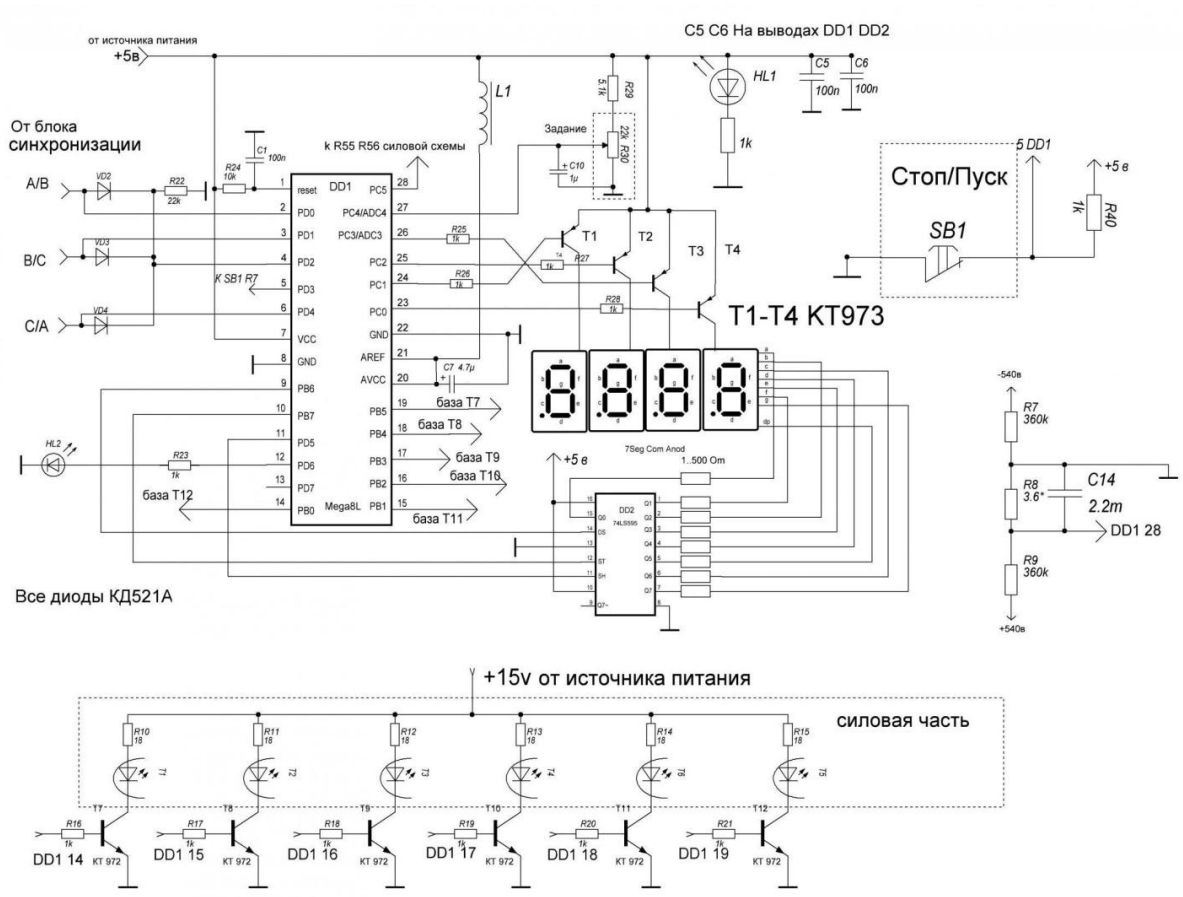

Рисунок 10 – Блок управления трехфазного тиристорного управляемого выпрямителя

Для начала изучим работу этого блока управления трехфазного тиристорного выпрямителя. Блок управления управляется микроконтроллером ATmega8. ATmega8 подаёт импульсы управления на тиристоры, и предоставляет индикацию режимов работы выпрямителя. Транзисторы T7-T12 усиливают импульсы. Микроконтроллер питается от внутреннего генератора. Так же в схеме есть светодиодная индикация. Она управляется через анодные ключи T1-T4. Включение и выключение работает посредством переключателя SB1. Замкнутое положение старт, разомкнутое положение стоп. Выходное напряжение регулируется с помощью потенциометра R30. Резисторы R7, R9 используются в делителе выходного напряжения. Дроссель L1 используется для фильтрации напряжения на АЦП микроконтроллера.

Когда включается микроконтроллер, он проверяет наличие синхронизирующих импульсов и в каком положении находится ключ SB1. Когда появляются первые синхронизирующие импульсы, выполняется проверка чередования фаз, A-B-C. Если тест был успешным, то устройство готово к эксплуатации, при этом на индикаторе будут гореть буквы «ABC», а светодиод HL1 будет светиться постоянно.

Если при включении микроконтроллера ключ SB1 разомкнут, то на индикаторе будет высвечиваться надпись «OFF», при этом HL1 будет мерцать с низкой частотой.

Рисунок 11 - Микроконтроллер ATmega8

Тест-кейсы для тестирования блока управления трехфазного тиристорного управляемого выпрямителя

Таблица 1 – Тест-кейс «Проверка питания микроконтроллера»

Шаг	Ожидаемый результат
Разомкнуть ключ SB1	Ключ был размокнут
Подать питание на микроконтроллер	Питание было подано, микроконтроллер включился
Поверить температуру микроконтроллера	Температура микроконтроллера находятся в рабочем диапазоне

Таблица 2 – Тест-кейс «Проверка работы светодиодной индикации»

Шаг	Ожидаемый результат
Разомкнуть ключ SB1	Ключ был размокнут
Подать питание на микроконтроллер	Питание было подано, микроконтроллер включился
Проверить работу светодиода HL1 при разомкнутом ключе SB1	Светодиод HL1 мерцает с низкой частотой
Замкнуть ключ SB1	Ключ SB1 был замкнут
Проверить работу светодиода HL1 при замкнутом ключе SB1	Светодиод HL1 горит постоянно

Таблица 3 – Тест-кейс «Проверка работы индикатора»

Шаг	Ожидаемый результат
Разомкнуть ключ SB1	Ключ был размокнут
Подать питание на микроконтроллер	Питание было подано, микроконтроллер включился
Проверить, что на индикаторе показывается надпись «OFF»	На индикаторе показывается надпись «OFF»
Замкнуть ключ SB1	Ключ SB1 был замкнут
Проверить, что синхронизирующие импульсы начали появляться	Синхронизирующие импульсы появились
Проверить, что на индикаторе показывается надпись «ABC»	На индикаторе показывается надпись «ABC»

Таблица 4 – Тест кейс «Проверка работы микроконтроллера»

Шаг	Ожидаемый результат
Разомкнуть ключ SB1	Ключ был размокнут
Подать питание на микроконтроллер	Питание было подано, микроконтроллер включился
Замкнуть ключ SB1	Ключ был замкнут
Проверить, что микроконтроллер проверяет наличие импульсов	Микроконтроллер проверяет наличие импульсов

Таблица 5 – Тест кейс «Проверка работы выпрямителя при отсутствии синхронизирующих импульсов более 7 секунд»

Шаг	Ожидаемый результат
Разомкнуть ключ SB1	Ключ был размокнут
Подать питание на микроконтроллер	Питание было подано, микроконтроллер включился
Замкнуть ключ SB1	Ключ SB1 был замкнут
Проверить, что синхронизирующие импульсы начали появляться	Синхронизирующие импульсы появились
Отключить подачу синхронизирующих импульсов более чем на 7 секунд	Синхронизирующие импульсы исчезли
Проверить информацию на индикаторе	Индикатор отображает цифры «380»
Возобновить подачу синхронизирующих импульсов	Синхронизирующие импульсы подаются
Проверить информацию на индикаторе	На индикаторе показывается надпись «АВС»

Заключение

В ходе проведенной научно-исследовательской работы были проанализированы все текущие направления развития тестирования. Так же, были составлены методологические указания для тестирования и диагностики управляемых полупроводниковых устройств и требования к системе отслеживания неисправностей и ошибок программного обеспечения для управляемых полупроводниковых устройств.

Список используемой литературы

1. **A. Bertolino.** Software Testing Research: Achievements, Challenges, Dreams / Antonia Bertolino – Istituto di Scienza e Tecnologiedell'Informazione "A. Faedo", 2007. – 200c
2. **L. Baresi.** Test oracles /L. Baresi, M. Young – Dept. of Comp. and Information Science, Univ. of Oregon, 2001. – 164c.
3. **E. Bayse.** A passive testing approach based on invariants: application to the wap. Computer Networks / E. Bayse, A. R. Cavalli. 2005. – 245c.
4. **Beizer.** Software Testing Techniques (2nd ed.) / B. Beizer – Van Nostrand Reinhold Co. 1990. – 366c.
5. **Belinfante.** Tools for test case generation / A. Belinfante, L. Frantzen, and C. Schallhart. 2006. – 103c.
6. **S. Berner.** Observations and lessons learned from automated testing / G. Bernot, M. C. Gaudelи B. Marre – Proc. 27th Int. Conf. on Sw. Eng. 2005. – 579c.
7. **G. Bernot.** Software testing based on formal specifications: a theory and a tool / G. Bernot, M. C. Gaudelи B. Marre – Softw. Eng. J., 6(6). 1991. – 405c.
8. **A. Bertolino.** ISSTA 2002 Panel: is ISSTA research relevant to industrial users? / A. Bertolino –Proc. ACM/SIGSOFT Int. Symp. on Software Testing and Analysis. 2002. – 202c.
9. **A. Bertolino.** Software testing / A. Bertolinoи E. Marchetti – Guide to the Software Engineering Body of Knowledge SWEBOK. 2004. – 168c.

10. **A. Bertolino.** Introducing a reasonably complete and coherent approach for modelbased testing / A. Bertolino, E. Marchettи H. Muccini – Electr. Notes Theor. Comput. 2005. – 256с.

11. **A. Bertolino.** The audition framework for testing web services interoperability / A. Bertolinoи A. Polini – Proc. EUROMICRO. 2005. – 103с.

12. **A. Bertolino.** Towards anti-model-based testing / A. Bertolino, A. Polini, P. Inverardiи H. Muccini – Proc. DSN 2004 (Ext. abstract). 2004. – 165с.

13. **S. Biffl.** Value-Based Software Engineering / S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, и P. Gruenbacher – Springer-Verlag, Heidelberg, Germany. 2006. – 196с.

14. **R. V. Binder.** Testing Object-Oriented Systems Models, Patterns, and Tools / R. V. Binder – Addison Wesley Longman, Inc. 2000. – 189с.

15. **C. Blundell.** Assume-guarantee testing / C. Blundell, D. Giannakopoulou, и C. S. Pasareanu – Proc. SAVCBS ACM Press. 2005. – 278с.

16. **G. V. Bochmann.** Protocol testing: review of methods and relevance for software testing / G. V. Bochmannи A. Petrenko – ACM/SIGSOFT Int. Symp. Software Testing and Analysis. 1994. – 194с.

17. **M. Boshernitsan**. From Daikon to Agitator: lessons and challenges in building a commercial tool for developer testing / M. Boshernitsan, R. Doong, and A. Savoia – ACM/SIGSOFT Int. Symp. Software Testing and Analysis, ACM Press. 2006. – 213с.

18. **L. Briand.** An investigation of graph-based class integration test order strategies / L. Briand, Y. Labiche, и Y. Wang – IEEE Trans. Softw. Eng. 2003. 607с.

19. **L. Briand.** Future of Software Engineering / L. Briand и A. Wolf – IEEE-CS Press. 2007. – 360с.

20. **L. C. Briand.** Automated, contract-based user testing of commercial-off-theshelf components / L. C. Briand, Y. Labicheи M. M. Sowka – 28th Int. Conf. on Sw. Eng., ACM Press. 2006. – 103с.

21. **M. Broy.** Model-Based Testing of Reactive Systems / M. Broy, B. Jonsson, J.-P. Katoen, M. Leuckerи A. Pretschner – Springer Verlag. 2005. – 263с.

22. **G. Canfora.** Testing services and servicecentric systems: Challenges and opportunities / G. Canfora и M. Di Penta – IT Professional. 2006. – 436с.

23. **N. Delgado.** A taxonomy and catalog of runtime software-fault monitoring tools / N. Delgado, A. Q. Gates и S. Roach – IEEE Trans. Softw. 2004. – 124с.

24. **S. Elbaum.** Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact / S. Elbaumи G. Rothermel– EmpiricalSoftw. Eng. 2005. – 435с.

25. **S. Elbaum.** Carving differential unit test cases from system test cases / S. Elbaum, H. N. Chin, M. B. Dwyer и J. Dokulil – 14th ACM/SIGSOFT Int. Symp. on Foundations of SwEng. 2006. – 264с.

26. **S. Elbaum.** Profiling deployed software: Assessing strategies and testing opportunities / S. Elbaumи M. Diep – IEEE Trans. Softw. Eng. 2005. – 265с.

27. **P. Frankl.** Provable improvements on branch testing / P. Frankl и E. Weyuker – IEEE Trans. Softw. Eng. 1993. – 975с.

28. **L. Frantzen.** Towards modelbased testing of web services / L. Frantzen, J. Tretmansи R. d. Vries – Int. Workshop on Web Services. 2006. – 82с.

29. **L. Frantzen.** A symbolic framework for model-based testing / L. Frantzen, J. Tretmansи T. Willemse – FATES/RV, LNCS 4262. 2006. – 236с.

30. **M. Gaudel.** Formal methods and testing: Hypotheses, and correctness approximations / M. Gaudel – FM 2005, LNCS 3582, Springer-Verlag. 2005. – 175с.

31. ГОСТ 2.701-84. Единая система конструкторской документации. Схемы виды и типы. Общие требования к выполнению. - Введ. 1985-07-01. - М.: Госстандарт СССР: Изд-во стандартов, 1984.- 11с.: ил.

32. ГОСТ 2.106-96. Единая система конструкторской документации. Текстовые документы. - Введ. 1997-07-01. - М.: Госстандарт РФ: Изд-во стандартов, 1996.- 39с.: ил.

33. ГОСТ 2.701-84. Единая система конструкторской документации. Правила выполнения схем.- Введ. 1985-07-01. - М.: Госстандарт СССР: Изд-во стандартов, 1994.- 11с.: ил.