Тестирование ПО

(продолжение)

Тестирование ПО

Метод функциональной декомпозиции

Главный принцип метода заключается в приведении всех тестовых случаев к некоторым функциональным задачам, которыми могут быть:

Тестирование ПО

– навигация (например, доступ к странице заказа из главного меню);

– бизнес-функции (например, оформление заказа);

– проверка данных (например, проверка состояния заказа);

– возврат к навигации.

Таким образом, самый верхний уровень скриптов представляет собой скрипт, содержащий серии вызовов одного или нескольких скриптов для конкретных тестовых случаев. Скрипты этих тестовых случаев вызывают необходимые скрипты бизнес-логики. Скрипты утилит могут вызываться в любом месте скрипта, где это необходимо.

Тестирование ПО

Метод Data-driven

Метод Data-driven (англ., Data Driven Testing, DDT, тестирование, управляемое данными) является продолжением метода функциональной декомпозиции.

Отличие состоит в том, что данные для тестов выносятся за пределы кода скрипта, как правило, в Excel-таблицу.

Например, для проверки авторизации может быть создана таблица, которая в каждой строке хранит имя пользователя, пароль и ожидаемый результат.

Тогда скрипт обращается к таблице с тестовыми

Тестирование ПО

данными и поочередно начинает их перебирать, выполняя нужные действия теста и сравнивая фактические результаты с ожидаемыми.

На основании проверок формируется журнал о прохождении тестов.

Метод Keyword-driven

Метод Keyword-driven или тестирование, управляемое ключевыми словами, можно рассматривать как продолжение и модификацию метода Data-driven. Тестовый сценарий здесь представлен в виде электронной таблицы, содержащей в одной из колонок специальные

Тестирование ПО

ключевые слова, а в остальных колонках могут быть тестовые данные, ожидаемые результаты и другая необходимая информация.

Ключевое слово является, как правило, функцией некоторой библиотеки и реализует предписанную последовательность действий.

Управляющий скрипт обращается к таблице тестового сценария, находит ключевое слово в нем, считывает из той же строки тестовые данные и выполняет действие, описанное этим ключевым словом. Полученный результат далее сравнивается с ожидаемым, и генерируется отчет о прохождении теста.

Тестирование ПО

Затем скрипт находит следующее ключевое слово, и так до конца документа. Таким образом, реализуется трехслойная модель автоматизации:

– 1-й слой – библиотека ключевых слов;

– 2-й слой – тестовый сценарий в виде электронной таблицы;

– 3-й слой – управляющий скрипт.

К преимуществам такого подхода можно отнести следующие:

1) тестовые сценарии могут быть реализованы в формате электронной таблицы, которая содержит все данные для ввода и проверки, таким образом, сценарий пишется только один раз;

Тестирование ПО

2) тестовые сценарии могут быть написаны в любом формате, который поддерживает конвертацию tab-delimited или comma-delimited;

3) создавать тестовые сценарии может любой тестировщик, даже не владеющий навыками автоматизированного тестирования;

4) если тестовые сценарии уже существуют в каком-либо формате, их не сложно перевести в формат электронных таблиц;

5) скрипты-утилиты, созданные для одного приложения, могут быть использованы с небольшими изменениями для тестирования других программ.

Тестирование ПО

Тесты, управляемые объектами (object-driven).

Тесты, управляемые объектами (Object-driven Testing, ODT) – это методология, при использовании которой тестируемое приложение в тестах представлено в виде класса (или нескольких классов). Для выполнения того или иного действия в приложении необходимо вызывать различные методы этого класса.

Некоторые инструменты автоматизации изначально разработаны под эту методологию. Ярким примером такого инструмента является SilkTest, в котором приложение представлено в виде winclass'ов и экземпляров этих классов – window.

Тестирование ПО

Другие инструменты могут косвенно поддерживать эту методологию благодаря возможностям языка. Например, в TestComplete есть два способа: использование специального объекта ODT или использование языковых возможностей.

Конечно, в случае ODT подхода никто не запрещает нам использовать одновременно обычные функции, не являющиеся методами класса. ODT всего лишь помогает представить тестируемое приложение и работу с ним в более удобном виде.

Объектами в ODT подходе могут выступать не только тестируемые приложения, но и другие сущности.

Тестирование ПО

При тестировании веб-приложений Object-driven testing зачастую называют подходом, основанным на Page Object'ах. При этом каждая страница веб-приложения описывается отдельным классом со своими свойствами и методами, поэтому по сути это одно и то же.

Тесты, управляемые моделями (model-based)

Тесты, управляемые моделями (Model-based Testing, MBT) – это отдельная методология, стоящая особняком в ряду подходов к автоматизации тестирования.

Тестирование ПО

В MBT используются совершенно другие инструменты и техники, на эту тему написано множество статей, и в рамках лекции мы дадим лишь краткое описание этого подхода.

Фактически MBT – это наиболее правильный подход к функциональному тестированию, так как он позволяет провести наиболее полное тестирование функциональности. Вместе с тем, MBT подход является наиболее трудоёмким.

Модель – это математическое описание приложения, описывающее его поведение и/или процесс тестирования приложения. В зависимости от сложности модели, мы можем провести более или менее точное тестирование приложения.

Тестирование ПО

При разработке моделей необходимо хорошо разбираться в функциональности тестируемого приложения и архитектуре компьютера. Например, работа с целыми числами и числами с плавающей запятой на компьютере выполняется по-разному, однако с точки зрения математики сложение и вычитание целых чисел является всего лишь подмножеством аналогичных операций с дробными числами.

Выбор точности модели обычно зависит от тестируемого приложения. Например, высокая точность очень важна в финансовых приложениях и жизненно необходима в медицинских, однако ею можно пренебречь в некоторых других случаях.

Тестирование ПО

В модель приложения также могут включаться пути тестирования. Например, процесс инсталляции приложения может включать в себя несколько экранов (папка для установки, устанавливаемые модули, лицензионное соглашение и т.д.). При этом пользователь может двигаться по экранам не только вперёд, но и назад вплоть до самого первого экрана, при этом рассчитывая, что ему не придётся затем заново вводить все данные (или повторно читать 10 страниц лицензионного соглашения). Поэтому для подобных случаев необходимо предусматривать сохранение состояния окна или страницы, чтобы в дальнейшем иметь возможность проверить все данные.

Тестирование ПО

Даже в случае простых приложений у нас появляется огромное количество вариантов, с которыми необходимо как-то работать, поэтому подход Model-based применяется обычно для приложений, к которым предъявляются повышенные требования качества. И именно поэтому для этих целей разработаны специальные инструменты.

Тем не менее, некий простой вариант Model-based тестирования можно реализовать и обычными инструментами, которые обычно используются при регрессионном тестировании, хотя процесс этот будет довольно трудоёмким.

Тестирование ПО

Для создания model-based тестов вам понадобится создать несколько сущностей, которые будут взаимодействовать между собой:

- Драйвер интерфейса – набор функций, осуществляющих взаимодействие с тестируемым приложением (ввод и получение данных).
- Хранилище состояния приложения – набор параметров, определяющих, в каком состоянии находится в данный момент приложение.
- Средства сравнения эталонных значений с полученными.
- Генератор тестовых данных. Набор функций, создающих разнообразные данные для выполнения тестирования.

Тестирование ПО

- Генератор тестов. Тесты в Model-based подходе являются абстрактными сущностями, т.е. они не завязаны на конкретные данные и/или пути выполнения приложения. Следовательно нам необходимо иметь возможность генерировать конкретные тесты из имеющихся абстрактных описаний.

Таким образом, структура подхода довольно сложная. Но, даже если вы не планируете использовать model-based тестирование в своем проекте, вы можете взять на вооружение некоторые из его подходов.

Тестирование ПО

Например, средства сравнения эталонных данных с полученными широко используются при тестировании функциональности CRUD (Create/Read/Update/Delete), - в этом случае вы один раз создаете в тесте структуру или запись, а затем заполняете поля формы или окна значениями из этой структуры, после чего открываете запись в приложении и сравниваете сохранённые данные с эталонными, хранящимися в созданной в самом начале структуре.

Тестирование ПО

Синхронизация выполнения тестов

Синхронизация – это, пожалуй, самый важный вопрос в функциональном тестировании, с которым необходимо разобраться независимо от того, какой инструмент вы используете или какой тип приложения тестируете. В зависимости от того, насколько эффективно вы будете использовать синхронизацию, настолько ваши тестовые скрипты будут стабильными и эффективными.

Под синхронизацией понимается задержка выполнения скрипта до тех пор, пока не произойдет определенное событие (например, появится сообщение или завершится загрузка страницы).

Тестирование ПО

В тестах необходимо предусматривать, где именно может потребоваться добавить синхронизацию для того, чтобы скрипт стабильно работал.

Синхронизация нужна для того, чтобы скрипты могли работать при разных условиях, которые влияют на скорость работы приложения (быстродействие компьютера, скорость сетевого подключения, загрузка удалённой базы данных и т.п.). Нужно всегда помнить, что написание и отладка скриптов тестировщиком могут проводиться на более быстром компьютере, чем компьютер, на котором в дальнейшем они будут выполняться. То же самое касается и других аспектов, таких как скорость сетевого подключения или загруженность СУБД.

Тестирование ПО

Например, представим такую ситуацию. Вы создаёте в приложении новую запись и нажимаете кнопку «Сохранить», после чего на экране появляется сообщение «Данные сохранены» с единственной кнопкой «ОК». Чтобы продолжить работу с приложением, необходимо закрыть окно, нажав на эту кнопку.

Допустим, что при написании теста вы используете локальную базу данных с единственным подключением (своим). Скорость доступа к данным при этом будет практически мгновенной, поэтому данные будут сохранены быстро и сообщение появится через полсекунды.

Тестирование ПО

Вы написали скрипт, который выполняет все необходимые действия и выложили его на тестовый сервер, где он будет запускаться вместе с другими тестами, но при этом тестируемое приложение будет использовать не локальную базу данных, а базу данных, расположенную на другом континенте.

Написанный вами скрипт подождёт полсекунды появления сообщения и выдаст ошибку «Сообщение не появилось», после чего сделает скриншот экрана и остановит выполнение. Однако на момент создания скриншота сообщение уже появится, что вы и увидите в логе теста.

Тестирование ПО

Отловить такие ошибки бывает очень трудно, особенно если для написания и регулярного запуска тестов используются разные окружения, более того, они могут проявляться лишь в определенные моменты времени, когда вы спите дома, а скрипты работают.

Именно для того, чтобы избежать подобных ситуаций, и используется синхронизация. Вместо того, чтобы пытаться сразу работать с окном, страницей или элементом управления, мы сначала ждём, пока этот элемент станет доступным.

Самый простой способ ожидания – это использование функций типа Sleep() или Delay(), которые просто ждут заданное время.

Тестирование ПО

Однако это – самый неэффективный способ, так как каждый раз задержки будут одинаковые, хотя ожидаемое событие может произойти как раньше, так и позже. В первом случае мы просто потеряем время на ненужное ожидание, во втором – получим ту же самую ошибку, какую получили бы без использования задержки, после чего придется еще больше увеличить ожидание, тем самым увеличивая время выполнения скрипта.

Для более эффективного решения этой проблемы можно ждать появления окна, ждать окончания загрузки страницы, ждать пока какое-то свойство элемента не станет равным определенному значению, ждать окончания работы процесса и т.д.

Тестирование ПО

Т.е. надо привязаться к какому-то событию в тестируемом приложении, которое нам даст сигнал, что можно продолжать.

Функции ожидания события и другие Wait-методы существуют во многих инструментах автоматизации, если же такого нет – их нужно обязательно написать самим, иначе будет очень трудно писать эффективные автоматизированные скрипты. Чтобы написать подобную функцию, можно, например, в цикле проверять свойство Exists элемента и возвращать True, если Exists=True, или False при выходе из цикла.

Здесь необходимо упомянуть о нескольких важных моментах.

Тестирование ПО

Во-первых, ожидая какое-то событие, не делайте ожидание бесконечным. Обязательно предусматривайте возможность выхода по таймауту, т.е. по прошествии определенного времени, так как приложение может элементарно зависнут, соответственно зависнет и скрипт.

Во-вторых, старайтесь при ожидании элементов не прописывать время ожидания жёстко. Лучше всего определить глобальную переменную для всего проекта и использовать её. В случае, если в дальнейшем вам понадобится увеличить или уменьшить время ожидания для всех операций, вы сделаете это в одном месте, а не сотнях или тысячах строчках кода.

Тестирование ПО

В-третьих, если вам придется самим писать функции ожидания, предусматривайте небольшие задержки в цикле между проверками, иначе цикл будет выполняться очень часто, занимая всё процессорное время и отбирая его у других приложений. При этом другие приложения, в свою очередь, будут работать медленнее.

Тестирование ПО

Семейство Selenium и его возможности

В рамках семейства Selenium имеется несколько программных продуктов, которые могут быть использованы в процессе функционального тестирования.

Например, Selenium Server позволяет организовать удаленный запуск браузера, при помощи Selenium Grid можно построить кластер из Selenium-серверов.

Также имеется «рекордер» Selenium IDE, который умеет записывать действия пользователя и генерировать код.

Тестирование ПО

Однако главным в семействе является Selenium WebDriver, который представляет собой библиотеку, позволяющую разрабатывать программы, управляющие поведением браузера.

Для группировки и запуска тестов, а также для генерации отчетов о тестировании при таком подходе используются фреймворки тестирования. Например, JUnit или TestNG для Java, NUnit или MSTest для .Net, RSpec или Cucumber для Ruby. Разработка тестов ведется, соответственно, в средах Eclipse, IntellijIDEA, MS Visual Studio, RubyMine и так далее.

Тестирование ПО

С целью следования принципам повторного использования тестового кода и функциональной декомпозиции, программный каркас для автоматизации тестирования с использованием Selenium WebDriver разделяется на несколько слоев (рис. 15.2): слой тестов, бизнес-слой, слой для работы с пользовательским интерфейсом и слой вспомогательных библиотек.

Слой тестов содержит описания сценариев в программном коде.

Бизнес-слой описывает наиболее часто используемые сложные операции, которые специфичны для тестируемой системы.

Тестирование ПО

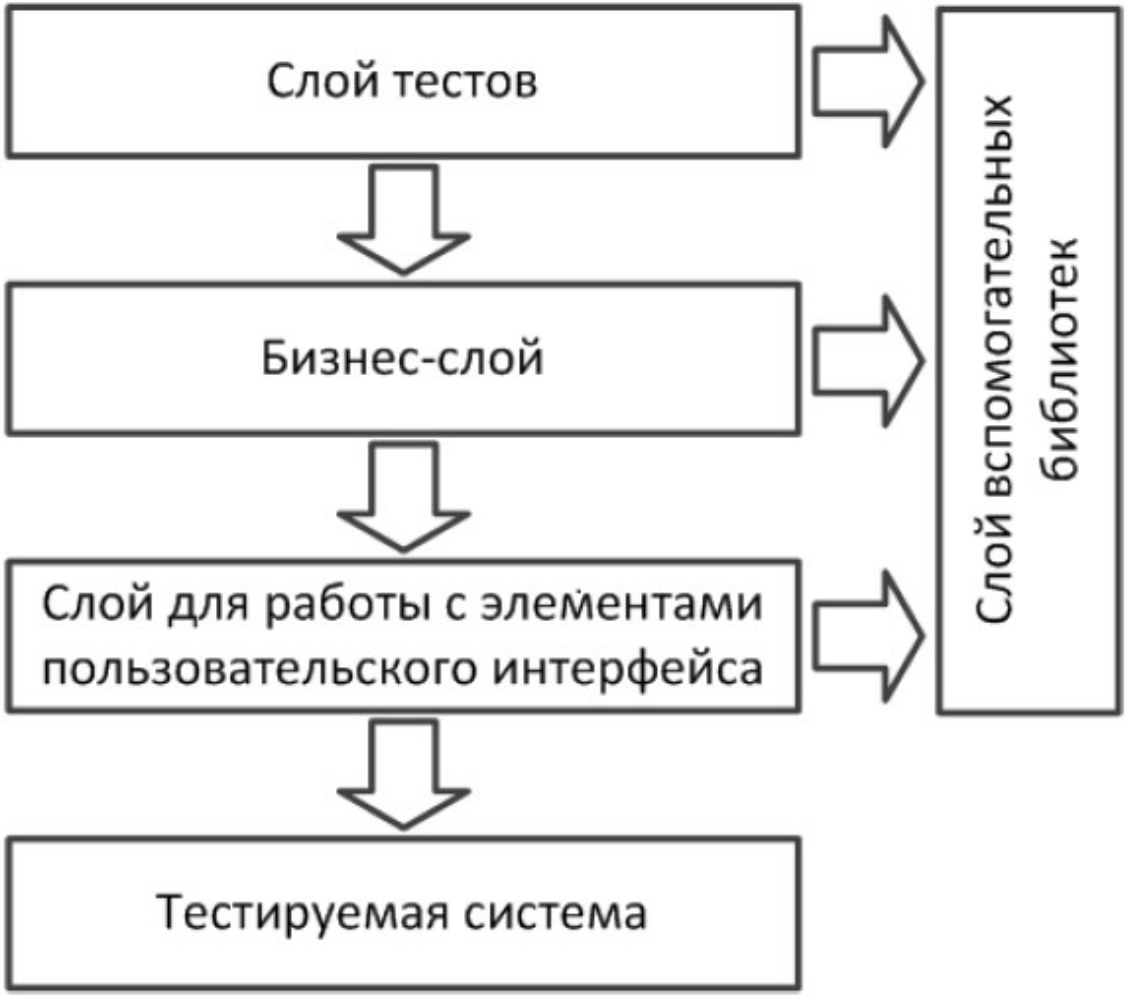

Рис. 15.2 - Схема каркаса для автоматизации тестирования с использованием Selenium WebDriver

Тестирование ПО

Слой для работы с элементами пользовательского интерфейса реализует взаимодействие тестирующего кода с тестируемой системой через пользовательский интерфейс. Данный слой в случае веб-приложений можно реализовать на основе программного интерфейса WebDriver.

Проблемы внедрения автоматизации тестирования

В ходе внедрения автоматизации тестирования существует риск не возврата вложений в этот процесс.

Тестирование ПО

При внедрении автоматизации необходимо рассчитать, насколько автоматизация эффективней, чем ручное тестирование.

Также необходимо учитывать следующие факторы:

– технологии, на которых построена система, а именно, наличие инструментов для их автоматизации;

– потенциальную изменяемость интерфейсов системы, на которых будет основываться автоматизация;

– специфику проверок системы, а именно, их пригодность для автоматизации;

– потенциальный объем автоматизируемых тестов;

Тестирование ПО

– планируемую частоту проведения тестирования;

– необходимость разработки специализированных инструментов для автоматизации;

– рост затрат на поддержку автоматизированных тестов;

– сроки разработки.

При оценке объема автоматизации и планируемого покрытия тестами, можно прибегнуть к разбиению тестов по уровням работы с системой и по видам тестирования.

Проблема внедрения автоматизации в короткие сроки решается использованием существующих программных каркасов для автоматизации

Тестирование ПО

тестирования, перенесением опыта автоматизации с других проектов, внедрением процесса разработки на основе тестов или на основе поведения.

Методика отладки

Отладка – это процесс обнаружения причин возникновения ошибок и их последующего исправления.

Отладка на некоторых проектах занимает до 50% времени разработки. Это одна из самых сложных частей программирования. По психологическим причинам отладка – это самая нелюбимая работа программиста.

Тестирование ПО

Многие программисты стремятся как можно быстрее написать программу, а затем обнаружить ошибки путем многократного ее выполнения с разнообразными тестовыми данными без их внимательного анализа и тщательного проектирования.

Это приводит к значительным затратам времени при установлении причин ошибок и их локализации, заметному снижению надежности программ, что в дальнейшем проявляется на этапе сопровождения программного изделия. Кроме того, известным фактом является внесение новых ошибок при отладке.

Тестирование ПО

В методике отладки принято выделять две части:

– нахождение причины возникновения ошибки (90% времени отладки, как правило, уходит на эту часть);

– исправление причины ошибки.

Наиболее эффективным считается обобщенный научный подход нахождения причин ошибок. Он состоит из следующих шагов:

1) сбор данных через повторяющиеся эксперименты;

2) создание гипотезы, отражающей максимум доступных данных;

3) разработка эксперимента для проверки гипотезы;

4) подтверждение или опровержение гипотезы;

Тестирование ПО

5) итерация предыдущих шагов (при необходимости).

Тогда, с учетом обобщенного научного подхода, методику отладки можно описать в виде последовательности следующих этапов:

1. Стабилизация ошибки (1-й шаг обобщенного научного подхода).
2. Обнаружение точного места ошибки (2–4 шаги обобщенного научного подхода).
3. Исправление ошибки.
4. Проверка исправления.
5. Поиск похожих ошибок.

Тестирование ПО

При стабилизации ошибки рекомендуется свести тест к минимально возможному. Если проблема возникает не стабильно, то ее практически невозможно диагностировать. Однако статистика показывает, что нестабильные ошибки обычно связаны с проблемами инициализации или висячими указателями.

В качестве рекомендаций по обнаружению точного места ошибки можно предложить следующие:

– используйте все доступные данные при выдвижении гипотез;

– воспроизведите ошибку различными способами;

Тестирование ПО

– используйте результаты негативных тестов;

– проведите «мозговой штурм» в поисках новых гипотез;

– сужайте подозрительные фрагменты кода;

– внимательнее отнеситесь к тем методам и функциям, где уже встречались ошибки;

– проверяйте недавние исправления;

– проводите интеграцию постепенно;

– ищите типичные ошибки;

– обсудите проблему с коллегами;

– отдохните от проблемы.

Тестирование ПО

Когда месторасположение ошибки определено, можно переходить к исправлению.

Здесь рекомендации могут быть следующими:

– разберитесь в проблеме, прежде чем ее исправлять;
– разберитесь в программе, а не только в проблеме;
– убедитесь в правильности диагноза до исправления;
– не торопитесь при внесении исправлений;
– сохраняйте исходную версию кода;
– исправляйте проблему, а не ее симптом;
– вносите исправления только тогда, когда вы уверены;
– вносите исправления только по одной штуке за раз;

Тестирование ПО

– проверяйте свои исправления;

– ищите похожие ошибки.

Методы отладки

Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации.

При этом используют различные методы:

– ручного тестирования;

– индукции;

Тестирование ПО

– дедукции;

– обратного прослеживания.

Метод ручного тестирования (также встречается название метод «грубой силы») – это самый простой и естественный способ, наиболее распространенный и традиционно используемый программистами. Всесторонний анализ за столом исходного кода и алгоритма программы, выходных результатов и сообщений компилятора, выполнение тестируемой программы вручную, используя тестовый набор, при работе с которым была обнаружена ошибка.

Для повышения эффективности отладки в текст программы включают операторы отладочного кода,

Тестирование ПО

которые регистрируют результаты использования конкретного оператора. Довольно часто проводят распечатку выборочных значений переменных.

После завершения отладки отладочные операторы можно оставить в виде комментариев для дальнейшего использования на этапе сопровождения.

Данный метод эффективен, однако трудно применим для больших программ, программ со сложными вычислениями, а также в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

Тестирование ПО

Метод индукции основан на тщательном анализе симптомов ошибки, используя стратегию движения от частного к общему.

Если компьютер просто «зависает», то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя. Полученную таким образом информацию организуют и тщательно изучают, просматривая соответствующий фрагмент программы.

В результате этих действий выдвигают гипотезы об ошибках, каждую из которых проверяют. Если гипотеза верна, то детализируют информацию об ошибке, иначе – выдвигают другую гипотезу.

Тестирование ПО

Организуя данные об ошибке, целесообразно записать все, что известно о ее проявлениях, причем фиксируют как ситуации, в которых фрагмент с ошибкой выполняется нормально, так и ситуации, в которых ошибка проявляется.

Если в результате изучения данных никаких гипотез не появляется, то необходима дополнительная информация об ошибке. Дополнительную информацию можно получить, например, в результате выполнения схожих тестов. В процессе доказательства пытаются выяснить, все ли проявления ошибки объясняет данная гипотеза, если не все, то либо гипотеза не верна, либо ошибок несколько.

Тестирование ПО

Метод дедукции реализует формирование множества причин, которые могли бы вызвать данное проявление ошибки. Затем анализируются причины и исключаются те, которые противоречат имеющимся данным.

Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента.

В противном случае наиболее вероятную гипотезу пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе – проверяют следующую причину.

Тестирование ПО

Метод обратного прослеживания (или инверсное прослеживание логики программы) наиболее эффективен для небольших программ и направлен на анализ логики выполнения программы в обратном направлении.

Отладка начинается с точки программы, где обнаружен неверный ожидаемый результат. На основе полученных в этой точке значений переменных, необходимо определить, исходя из логики программы, какие результаты должны были быть при правильной работе программы.

Последовательное продвижение к началу программы позволяет достаточно быстро и точно определить место (и причину возникновения)

Тестирование ПО

ошибки, т. е. место между оператором, где результат выполнения программы соответствовал ожидаемому, и оператором, в котором появились расхождения. Процесс продолжают, пока не обнаружат причину ошибки.

Средства отладки

Большинство современных сред программирования включают средства отладки, которые обеспечивают максимально эффективную отладку. Они позволяют:

– выполнять программу по шагам, причем как с заходом в подпрограммы, так и выполняя их целиком;

Тестирование ПО

– предусматривать точки останова;

– выполнять программу до оператора, указанного курсором;

– отображать содержимое любых переменных при пошаговом выполнении;

– отслеживать поток сообщений и т. п.

Применять интегрированные средства отладки в рамках среды достаточно просто. Используют разные приемы в зависимости от проявлений ошибки.

Если получено сообщение об ошибке, то сначала уточняют, при выполнении какого оператора программы оно получено.

Тестирование ПО

Для этого устанавливают точку останова в начало фрагмента, в котором проявляется ошибка, и выполняют операторы в пошаговом режиме до проявления ошибки.

Если получены неверные результаты теста, то локализовать ошибку обычно сложнее. В этом случае сначала определяют фрагмент, при выполнении которого получаются неправильные результаты.

Для этого последовательно проверяют интересующие значения в узловых точках. Обнаружив значения, отличающиеся от ожидаемых, по шагам трассируют соответствующий фрагмент до выявления оператора, выполнение которого дает неверный результат.

Тестирование ПО

Для уточнения природы ошибки возможен анализ машинных кодов, флагов и представления программы и значений памяти в шестнадцатеричном виде.

Кроме отладчиков можно выделить следующие средства отладки:

1. Специальные средства расширения языка программирования для контроля типов и диапазонов значений данных, обработки исключительных ситуаций и т.д.
2. Средства для печати значений используемых переменных при аварийном завершении программы.

Тестирование ПО

3. Пакеты программ для прослеживания потоков управления и данных в программе, контроля индексов и регистрации вызовов программ.
4. Генераторы тестовых данных, формирующие требуемые тестовые наборы.
5. Специальные онлайн-отладчики, обеспечивающие автоматизацию рестартов, остановов и прерываний программы, просмотр работы отдельных операторов и т. п.
6. CASE-средства для построения, например, схем потоков данных, модулей данных, схем алгоритмов.

Тестирование ПО

7. Автоматизированные рабочие места программистов, включающие большинство из перечисленных средств.