**Тестирование и отладка**

**Выполнил студент группы 4ПК2**

**Кузнецов Илья Владиславович**

**Тестирование - это испытание, проверка правильности работы программы в целом, либо её составных частей. В процессе тестирования проверяется работоспособность программы, не содержащей явных ошибок**. Как бы ни была тщательно отлажена программа, решающим этапом, устанавливающим ее пригодность для работы, является контроль программы по результатам ее выполнения на системе тестов. **Программу условно можно считать правильной, если её запуск для выбранной системы тестовых исходных данных во всех случаях дает правильные результаты.**

**Но, как справедливо указывал известный теоретик программирования Э. Дейкстра, тестирование может показать лишь наличие ошибок, но не их отсутствие. Нередки случаи, когда новые входные данные вызывают "отказ" или получение неверных результатов работы программы, которая считалась полностью отлаженной. Для реализации метода тестов должны быть изготовлены или заранее известны эталонные результаты. Вычислять эталонные результаты нужно обязательно до, а не после получения машинных результатов. В противном случае имеется опасность невольной подгонки вычисляемых значений под желаемые, полученные ранее на машине.**

**Отладка программы - это процесс поиска и устранения ошибок в программе, производимый по результатам её прогона на компьютере. При отладке происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования; В современных программных системах отладка осуществляется часто с использованием специальных программных средств, называемых отладчиками. Эти средства позволяют исследовать внутреннее поведение программы.**

**Программа-отладчик обычно обеспечивает следующие возможности:**

**- пошаговое исполнение программы с остановкой после каждой команды (оператора);**

**- просмотр текущего значения любой переменной или нахождение значения любого выражения, в том числе, с использованием стандартных функций;**

**- при необходимости можно установить новое значение переменной;**

**- установку в программе "контрольных точек", т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.**

**Верификацией и аттестацией называют процессы проверки и анализа, в ходе которых проверяется соответствие программного обеспечения своей спецификации и требованиям заказчиков. Верификация и аттестация охватывают полный жизненный цикл ПО – они начинаются на этапе анализа требований и завершаются проверкой программного кода на этапе тестирования готовой программной системы. Верификация и аттестация не одно и то же, хотя их легко перепутать. Кратко различие между ними можно определить следующим образом: верификация отвечает на вопрос, правильно ли создана система; аттестация отвечает на вопрос, правильно ли работает система.**

**Верификация проверяет соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям. Аттестация– более общий процесс. Во время аттестации необходимо убедиться, что программный продукт соответствует ожиданиям заказчика. Аттестация проводится после верификации, для того чтобы определить, насколько система соответствует не только спецификации, но и ожиданиям заказчика.**

**Как уже отмечалось ранее, на ранних этапах разработки ПО очень важна аттестация системных требований. В требованиях часто встречаются ошибки и упущения; в таких случаях конечный продукт, вероятно, не будет соответствовать ожиданиям заказчика. Но, конечно, аттестация требований не может выявить все проблемы в спецификации требований. Иногда недоработки и ошибки в требованиях обнаруживаются только после завершения реализации системы.**

**В процессах верификации и аттестации используются две основные методики проверки и анализа систем.**

**Инспектирование ПО. Анализ и проверка различных представлений системы, например документации спецификации требований, архитектурных схем или исходного кода программ. Инспектирование выполняется на всех этапах процесса разработки программной системы. Параллельно с инспектированием может выполняться автоматический анализ исходного кода программ и соответствующих документов. Инспектирование и автоматический анализ – это статические методы верификации и аттестации, поскольку им не требуется исполняемая система.**

**Тестирование ПО. Запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы. Тестирование – это динамический метод верификации и аттестации, так как применяется к исполняемой системе.**

**Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. 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;**

**- дедукции;**

**- обратного прослеживания.**

**Метод ручного тестирования. Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которым была обнаружена ошибка.**

**Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций.**

**Данный метод часто используют как составную часть других методов отладки.**

**Метод индукции. Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке. 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.**

**Метод дедукции. По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать.**

**Метод обратного прослеживания. Для небольших программ эффективно применение метода обратного прослеживания. Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Далее, исходя из этой гипотезы, делают предложения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.**

**Тестирование белого ящика — метод тестирования ПО, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику.**

**Тестирование серого ящика — метод тестирования ПО, который предполагает комбинацию White Box и Black Box подходов. То есть, внутреннее устройство программы нам известно лишь частично.**

**Тестирование чёрного ящика — также известное как тестирование, основанное на спецификации или тестирование поведения — техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы.**

**Уровни Тестирования: Компонентное или Модульное тестирование (Component or Unit Testing)**

**Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится вызывая код, который необходимо проверить и при поддержке сред разработки, таких как фреймворки (frameworks — каркасы) для модульного тестирования или инструменты для отладки. Все найденные дефекты, как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).**

**Один из наиболее эффективных подходов к компонентному (модульному) тестированию — это подготовка автоматизированных тестов до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор, пока все тесты не будут успешно пройдены.**

**Разница между компонентным и модульным тестированием:**

**По-существу, эти уровни тестирования представляют одно и тоже и разница лишь в том, что в компонентном тестировании, в качестве параметров функций, используют реальные объекты и драйверы, а в модульном тестировании — конкретные значения.**

**Интеграционное тестирование (Integration Testing)**

**Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).**

**Уровни интеграционного тестирования:**

**Компонентный интеграционный уровень ( Component Integration testing) проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.**

**Системный интеграционный уровень (System Integration Testing) — проверяется взаимодействие между разными системами после проведения системного тестирования.**

**Подходы к интеграционному тестированию:**

**Снизу вверх (Bottom Up Integration):**

**Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.**

**Сверху вниз (Top Down Integration):**

**Сначала тестируются все высокоуровневые модули, затем постепенно, один за другим, добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем, по мере готовности, они заменяются реальными активными компонентами. Таким образом, мы проводим тестирование сверху вниз.**

**Большой взрыв («Big Bang» Integration):**

**Все (или практически все) разработанные модули собираются вместе в виде законченной системы или ее основной части, а затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако, если тест-кейсы и их результаты записаны неверно, то сам процесс интеграции будет осложнен, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования.**

**Системное тестирование (System Testing)**

**Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований ,дефекты в системе в целом. При этом выявляется неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, во время тестирования рекомендуется использовать окружение, максимально приближенное к тому, на которое будет установлен продукт после выдачи.**

**Можно выделить два подхода к системному тестированию:**

**на базе требований (requirements based) — для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.**

**на базе случаев использования (use case based) — на основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся тест-кейсы (test cases), которые должны быть протестированы.**

**Приемочное тестирование или приемо-сдаточное испытание (Acceptance Testing)**

**Приемочное тестирование или приемо-сдаточное испытание (Acceptance Testing) — формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:**

**определения удовлетворения системой приемочным критериям;**

**вынесения решения заказчиком или другим уполномоченным лицом принятия приложения.**

**Приемочное тестирование выполняется на основании набора типичных тестовых случаев и сценариев, разработанных на основании требований к данному приложению.  
Решение о проведении приемочного тестирования принимается, когда:**

**Продукт достиг необходимого уровня качества.**

**Заказчик ознакомлен с Планом Приемочных Работ (Product Acceptance Plan) или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.**

**Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.**

**Системное тестирование программного обеспечения — это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований к системе в целом.   
При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня,   
несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.**

**Тестирование производительности ( Performance testing ).**

**Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:**

**Измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций.**

**Определение количества пользователей, одновременно работающих с приложением.**

**Определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций).**

**Исследование производительности на высоких, предельных, стрессовых нагрузках.**

**Стрессовое тестирование ( Stress Testing )**

**Стрессовое тестирование позволяет проверить, насколько приложение и система в целом работоспособны в условиях стресса, а также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию, после прекращения воздействия стресса. Стрессом, в данном контексте, может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также, одной из задач при стрессовом тестировании может быть оценка деградации производительности. Таким образом, цели стрессового тестирования могут пересекаться с целями тестирования производительности.**

**Объемное тестирование ( Volume Testing )**

**Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:**

**Измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций.**

**Может производиться определение количества пользователей, одновременно работающих с приложением.**

**Тестирование стабильности или надежности( Stability / Reliability Testing)**

**Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.**

**В англоязычной терминологии вы можете так же найти еще один вид тестирования — Load Testing — тестирование реакции системы на изменение нагрузки (в пределе допустимого). Нам показалось, что Load и Performance преследуют все же одну и ту же цель: проверка производительности (времени отклика) на разных нагрузках. Собственно поэтому мы и не стали разделять их. В то же время кто то может разделить. Главное все таки понимать цели того или иного вида тестирования и постараться их достигнуть.**

**Регрессионное тестирование — задача, с которой сталкивается каждый тестировщик. Ведь любой предмет после изменений в одном месте может начать ломаться в месте, где раньше работал исправно**

**Регрессионное тестирование (Regression testing) — Тестирование уже протестированной программы,  
проводящееся после модификации для уверенности в том, что процесс модификации не внес или не активизировал ошибки в областях, не подвергавшихся изменениям. Проводится после изменений в коде программного продукта или его окружении.**

**Регрессионное тестирование требуется, когда есть**

**Изменение требований и кода изменяется в соответствии с требованием**

**Новая функция добавлена ​​в программное обеспечение**

**Устранение дефектов**

**Исправлена ​​проблема с производительностью**

**Инструменты регрессионного тестирования**

**Если ваше программное обеспечение подвергается частым изменениям, затраты на регрессионное тестирование будут расти.**

**В таких случаях ручное выполнение тестовых случаев увеличивает время выполнения теста, а также затраты.**

**Автоматизация регрессионных тестов является разумным выбором в таких случаях.**

**Степень автоматизации зависит от количества тестовых случаев, которые можно повторно использовать для последовательных циклов регрессии.**

**Ниже приведены наиболее важные инструменты, используемые для функционального и регрессионного тестирования в разработке программного обеспечения.**

[**Ranorex Studio**](https://translate.googleusercontent.com/translate_c?depth=1&pto=aue&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://bit.ly/2XolITW&usg=ALkJrhjR3nRUf7MZ0tCH37xhh2kXH7hnFA)**:**[**универсальная**](https://translate.googleusercontent.com/translate_c?depth=1&pto=aue&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://bit.ly/2XolITW&usg=ALkJrhjR3nRUf7MZ0tCH37xhh2kXH7hnFA)**автоматизированная система регрессионного тестирования для настольных, веб-и мобильных приложений со встроенным Selenium WebDriver. Включает в себя полный IDE плюс инструменты для автоматизации без кода.**

[**Selenium**](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-selenium/uchebnik-selenium)**: это инструмент с открытым исходным кодом, используемый для автоматизации веб-приложений. Selenium можно использовать для регрессионного тестирования на основе браузера.**

[**Quick Test Professional (QTP)**](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-qtp/uchebnik-qtp)**: HP Quick Test Professional — это автоматизированное программное обеспечение, разработанное для автоматизации функциональных и регрессионных тестов. Он используетязык[VBScript](https://coderlessons.com/tutorials/veb-razrabotka/uchebnik-vbscript/uchebnik-vbscript) для автоматизации. Это управляемый данными инструмент на основе ключевых слов.**

**Rational Functional Tester (RFT) : рациональный функциональный тестер IBM — это инструмент**[**Java**](https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/uchebnik-java/uchebnik-java)**, используемый для автоматизации тестовых случаев программных приложений. Это в основном используется для автоматизации регрессионных тестов, а также интегрируется с Rational Test Manager.**

**Регрессионное тестирование и управление конфигурацией**