Практическая работа

Тема: Основытестированияпрограммногообеспечения

Теоретическая часть

Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта.

С технической точки зрения тестирование заключается в выполнении приложения на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными) с целью установления соответствия различных свойств и характеристик приложения заказанным свойствам. Как одна из основных фаз процесса разработки программного продукта (дизайн приложения - Разработка кода - Тестирование) тестирование характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта. Широко известна оценка распределения трудоемкости между фазами создания программного продукта: 40-20-40 %, из чего следует, что наибольший эффект в снижении трудоемкости может быть получен, прежде всего, на фазах Дизайн приложения и Тестирование. Поэтому основные вложения в автоматизацию или генерацию кода следует осуществлять, прежде всего, на этих фазах.

Хотя в современном индустриальном программировании автоматизация тестирования является широко распространенной практикой, технология верификации требований и спецификаций пока делает только свои первые шаги. Задачей ближайшего будущего является движение в сторону такого распределения трудоемкости (60-20-20 %), чтобы суммарная цена обнаружения большинства дефектов стремилась к минимуму за счет обнаружения пре имущественного числа на наиболее ранних фазах разработки программного продукта.

Концепция тестирования

Программа-это аналог формулы в обычной математике.

Формула для функции f, полученной суперпозицией функций

f1,f2, ...,fn, - выражение, описывающее эту суперпозицию:

f= fl \*f2\*f3\*... \*fn.

Если аналог fl ,f2,f3...,\*fn - операторы языка программирования, то их формула - программа.

Существуют два метода обоснования истинности формул:

1. Формальный подход, или доказательство, применяется, когда из исходных формул-аксиом с помощью формальных процедур (правил вывода) выводятся искомые формулы и утверждения (теоремы). Вывод осуществляется путем перехода от одних формул к другим по строгим правилам, которые позволяют свести процедуру перехода от формулы к формуле к последовательности текстовых подстановок:

*А\*\*3 =А\*А\*А;*

*А\*А\*А =А R,A\*R R,A\*R R.*

Преимущество формального подхода заключается в том, что с его помощью удается избегать обращений к бесконечной области значений и на каждом шаге доказательства оперировать только конечным множеством символов.

2. *Интерпретационный подход* применяется, когда осуществляется подстановка констант в формулы, а затем - интерпретация формул как осмысленных утверждений в элементах множеств конкретных значений. Истинность интерпретируемых формул проверяется на конечных множествах возможных значений. Сложность подхода состоит в том, что на конечных множествах комбинации возможных значений для реализации исчерпывающей проверки могут оказаться достаточно велики.

Интерпретационный подход используется при экспериментальной проверке соответствия программы своей спецификации. Применение интерпретационного подхода в форме экспериментов над исполняемой программой составляет суть отладки и тестирования.

Терминология тестирования

*Отладка* (debug, debugging) - процесс поиска, локализации' и исправления ошибок в программе [IEEE Std.610-12.1990]. Термин «отладка» в отечественной литературе используется двояко: для обозначения активности по поиску ошибок (собственно *тестирование),* по нахождению причин их появления н исправлению или активности по локализации и исправлению ошибок.

*Тестирование* обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок).

Как правило, на *фазе тестирования* осуществляется и исправление идентифицированных ошибок, включающее локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы тестируемого приложения (Application Under Testing (AUT) или Implementation Under Testing (IUT)).

Если программа не содержит синтаксических ошибок (прошла трансляцию) и может быть выполнена на компьютере, она обязательно вычисляет какую-либо функцию, осуществляющую отображение входных данных в выходные. Это означает, что компьютер на своих ресурсах доопределяет частично определенную программой функцию до тотальной определенности. Следовательно, судить о правильности или неправильности результатов выполнения программы можно, только сравнивая спецификацию желаемой функции с результатами ее вычисления, что и осуществляется в процессе тестирования.

Тестирование разделяют на статическое идинамическое:

Статическое *тестирование* выявляет формальными методами анализа без выполнения тестируемой программы неверные конструкции или неверные отношения объектов программы (ошибки формального задания) с помощью специальных инструментов контроля кода - CodeChecker.

Динамическое *тестирование* (собственно тестирование) осуществляет выявление ошибок только на выполняющейся программе с помощью специальных инструментов автоматизации тестирования - Testbed или Testbench.

Организация тестирования

Организация тестирования ПО регулируется следующими стандартами:

- IEEE829-1998 *Standard for Software Test Documentation:*описывает виды документов, служащих для подготовки тестов.

*-*IEEE1008-1987 (Rl993, R2002) *Standard for Software Unit Testing:* описывает организацию модульного тестирования.

*-*ISOIIEC12119:1994 *(аналог АS/NZS* 4366:1996 *и ГОСТ* Р-2000, *также принят IEEE под номером IEEE* 1465-1998) *Information Technology. Software packages* - *Quality requirements and testing:* описывает требования к процедурам тестирования программных систем.

Тестирование осуществляется на заданном заранее множестве входных данных *Х* и множестве предполагаемых результатов *Y* - *(Х,* Y), которые задают график желаемой функции. Кроме того, зафиксирована процедура Оракул (oracle), которая определяет, соответствуют ли выходные данные - *Yв* (вычисленные по входным данным - Х) желаемым результатам - Y*,* т.е. принадлежит ли каждая вычисленная точка *(х,ув)* графику желаемой функции *(Х, Y*).

Оракул дает заключение о факте появления неправильной пары *(х,ув)* и ничего не говорит о том, каким образом она была вычислена или каков правильный алгоритм - он только сравнивает вычисленные и желаемые результаты. Оракулом может быть даже заказчик или программист, производящий соответствующие вычисления в уме, поскольку Оракулу нужен какой-либо альтернативный способ получения функции *(Х Y)* для вычисления эталонных значений Y*.*

Итак, в процессе тестирования сравнение промежуточных результатов с полученными независимо эталонными результатами позволяет найти причины и место ошибки, исправить текст программы, провести повторную трансляцию и настройку на выполнение и продолжить тестирование.

Тестирование заканчивается, когда выполнилось или «прошло» (pass) успешно достаточное количество тестов в соответствии · с выбранным критерием тестирования.

Тестирование - это:

- процесс выполнения ПО системы или компонента в условиях анализа или записи получаемых результатов с целью проверки (оценки) некоторых свойств тестируемого объекта;

- процесс анализа пункта требований к ПО с целью фиксации различии между существующим состоянием ПО и требуемым (что свидетельствует о проявлении ошибки) при экспериментальной проверке соответствующего пункта требований;

- контролируемое выполнение программы на конечном множестве тестовых данных и анализ результатов этого выполнения для поиска ошибок [IEEE Std 829-1983].

для

Реализация тестирования разделяется на три этапа (рис. 1.):

1. Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment).

2. Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver [IEEE Std 829-1983]) с получением протокола результатов тестирования (test log).

3. Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

Тестирование - конечная процедура. Набор ситуаций, в которых будет проверяться тестируемое ПО, всегда конечен. Более того, он должен быть настолько мал, чтобы тестирование можно было провести в рамках проекта разработки ПО, не слишком увеличивая его бюджет и сроки. Это означает, что при тестировании всегда проверяется очень небольшая доля всех возможных ситуаций.

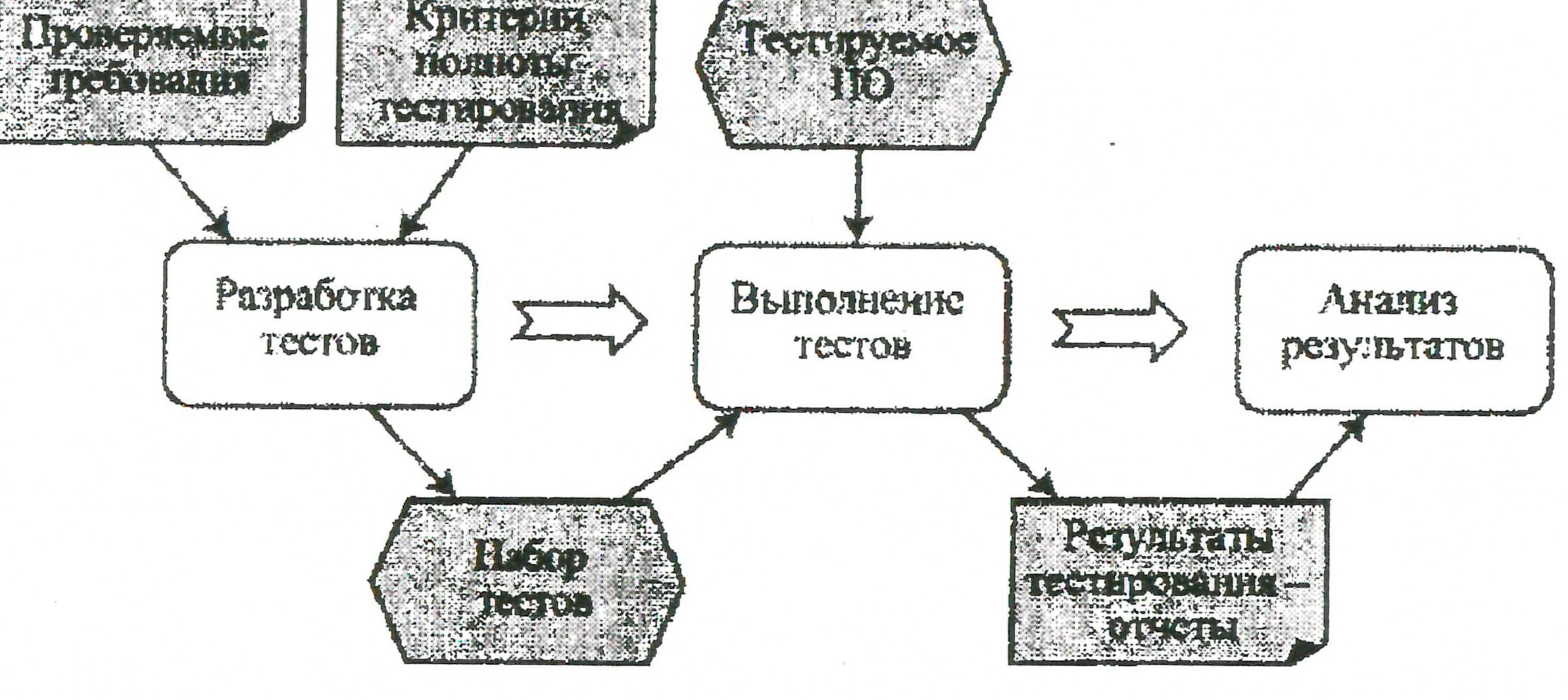


Рис. 1. Схема процесса тестирования

Основные задачи тестирования: построить такой набор ситуаций, который был бы достаточно представителен и позволял бы завершить тестирование с достаточной степенью уверенности в правильности ПО вообще, убедиться, что в конкретной ситуации ПО работает правильно, в соответствии с требованиями.

Основными проблемами тестирования являются:

1. Тестирование программы на всех входных значениях невозможно. Невозможно тестирование и на всех путях. Следовательно, надо отбирать конечный набор тестов, позволяющий проверить программу на основе наших интуитивных представлений.

2. Требование к тестам - программа на любом из них должна останавливаться, т.е. не зацикливаться. Можно ли заранее гарантировать останов на любом тесте? В теории алгоритмов доказано, что не существует общего метода для решения этого вопроса, а также вопроса, достигнет ли программа на данном тесте заранее фиксированного оператора.

3. Задача о выборе конечного набора тестов (Х. У) для проверки программы в общем случае неразрешима. Поэтому для решения практических задач остается искать частные случаи решения этой задачи.

Критериивыборатестов

Требования к идеальному критерию:

1. Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.

2. Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

3. Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы.

4. Критерий должен быть легко проверяемым, например вычисляемым на тестах.

Для нетривиальных классов программ в общем случае не существует полного и надежного критерия, зависящего от программ или спецификаций.

Поэтому мы стремимся к идеальному общему критерию через реальные частные.

Классы критериев следующие:

1. Структурные критерии используют информацию о структуре программы (критерии так называемого «белого ящика»).

2. Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого «черного ящика»).

3. Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения средствами проверки некоторой статистической гипотезы.

4. Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

Уровни тестирования

Различают следующие уровни тестирования:

Модульное тестирование (юнит-тестирование) - процесс, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволит достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, т.е. к появлению ошибок в уже написанных и оттестированных местах программы, а также облегчает локализацию и устранение таких ошибок. Цель юнит-тестирования - изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Этот тип тестирования обычно выполняется программистами.

Преимуществами модульного тестирования являются:

- поощрение изменений: юнит-тестирование позже позволяет программистам проводить рефакторинг с уверенностью, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений;

- упрощение интеграции: юнит-тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх» - сначала тестируются отдельные части программы, затем программа в целом;

- документирование кода: юнит-тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит­ тест в качестве примера;

- отделение интерфейса от реализации: поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним.

Интеграционное тестирование проверяет, есть ли какие-либо проблемы в интерфейсах и взаимодействии между интегрируемыми компонентами (например, не передается информация, передается некорректная информация). Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приемным и требованиям надежности. Тестирование этих проектируемых единиц - объединения, множества или группа модулей - выполняется через их интерфейс с использованием тестирования «черного ящика». При тестировании «черного ящика» тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру или другому процессу подключиться к системе для тестирования.

Системное тестирование - это тестирование программного обеспечения, выполняемое на полной, интегрированной системе с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования «черного ящика» и тем самым не требует знаний о внутреннем устройстве системы.

Альфа-тестирование и бета-тестирование являются подкатегориями системного тестирования. Альфа- и бета-тестирование относятся к стадиям до выпуска продукта.

Альфа-тестирование - имитация реальной работы с системой штатными разработчиками либо реальная работа с системой потенциальными пользователями/заказчиком на стороне разработчика. Часто альфа-тестирование применяется для законченного продукта в качестве внутреннего приемочного тестирования. Иногда альфа­тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщикам для дополнительного исследования в окружении, подобном тому, в котором будет использоваться ПО.

Бета-тестирование: в некоторых случаях выполняется распространение версии с ограничениями (по функциональности или времени работы) для некоторой группы лиц, с тем чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета­ тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей. Бета-тестирование в целом ограничено техникой «черного ящика» (хотя постоянная часть тестировщиков обычно продолжает тестирование «белого ящика» параллельно бета-тестированию). Таким образом, термин «бета­ тестирование» может указывать на состояние программы (ближе к выпуску, чем «альфа-тестирование») или может указывать на некоторую группу тестировщиков и процесс, выполняемый этой группой. При тестировании «белого ящика» (англ. white-box testing, также говорят - «прозрачного ящика») разработчик теста имеет доступ к исходному коду и может писать код, который связан с библиотеками тестируемого ПО. Это типично для юнит-тестирования (англ. Unit testing), при котором тестируются только отдельные части системы. Оно обеспечивает работоспособность и устойчивость компонентов и конструкции.

Юзабилити-тестирование - эксперимент, выполняемый с целью определения, насколько хорошо люди могут использовать некоторый искусственный объект (такой как веб-страниц пользовательский интерфейс или устройство) для его предполагаемого применения. Юзабилити-тестирование сосредоточено на определенном объекте или небольшом наборе объектов, в то время как исследования взаимодействия «человек - компьютер» в целом формулируют универсальные принципы. Юзабилити-тестирование - метод оценки удобства использования продукта, основанный на привлечении пользователей в качестве тестирующих.

Тестирование безопасности - оценка уязвимости программного обеспечения к различным атакам. Компьютерные системы очень часто являются мишенью незаконного проникновения. Под проникновением понимается широкий диапазон действий: попытки хакеров проникнуть в систему из спортивного интереса, месть рассерженных служащих, взлом мошенниками для незаконной наживы. Тестирование безопасности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение. В ходе тестирования безопасности испытатель играет роль взломщика. Ему разрешено все:

- попытки узнать пароль с помощью внешних средств;

- атака системы с помощью специальных утилит, анализирующих защиты;

- подавление системы (в надежде, что она откажется обслуживать других клиентов);

- целенаправленное введение ошибок в надежде проникнуть в систему в ходе восстановления;

- просмотр несекретных данных в надежде найти ключ для входа в систему.

Конечно, при неограниченном времени и ресурсах хорошее тестирование безопасности взломает любую систему. Задача проектировщика системы - сделать цену проникновения более высокой, чем цена получаемой в результате информации.

Тестирование производительности - оценка функции производительности программного обеспечения. Тестирование производительности проверяет скорость работы ПО в компьютерной системе. Производительность тестируется на всех шагах процесса тестирования. Даже на уровне элемента при проведении тестов «белого ящика» может оцениваться производительность индивидуального модуля. Тем не менее пока все системные элементы не объединятся полностью, не может быть установлена истинная производительность системы.

Иногда тестирование производительности сочетают со стрессовым тестированием. При этом нередко требуется специальный аппаратный и программный инструментарий. Например, часто требуется точное измерение используемого ресурса (процессорного цикла и т.д.). Внешний инструментарий регулярно отслеживает интервалы выполнения, регистрирует события (например, прерывания) и машинные состояния. С помощью инструментария испытатель может обнаружить состояния, которые приводят к деградации и возможным отказам системы.

Регрессионное тестирование - собирательное название для всех видов тестирования программного обеспечения, направленных на обнаружение ошибок в уже протестированных участках исходного кода. Такие ошибки (когда после внесения изменений в программу перестает работать то, что должно было продолжать работать) называют регрессионными ошибками.

Регрессионное тестирование является неотъемлемой частью экстремального программирования. В этой методологии проектная документация заменяется на расширяемое, повторяемое и автоматизированное тестирование всего программного пакета на каждой стадии цикла разработки программного обеспечения. Регрессионное тестирование может быть использовано не только для проверки корректности программы, часто оно также используется для оценки качества полученного результата. Так, при разработке компилятор при прогоне регрессионных тестов рассматриваются размер полу чаемого кода, скорость его выполнения и время компиляции каждого из тестовых примеров.

Автоматическое тестирование использует программное обеспечение для проверки выполнения проводимых тестов, что помогает сократить время тестирования и упростить его процесс.

Современные средства разработки создают довольно сложные приложения, и их ручное тестирование является очень трудоемким процессом. Недостаток ручного тестирования заключается также в том, что результаты выполнения тестов не сохраняются и их трудно повторить заново. Автоматические тесты позволяют упростить процесс ручного тестирования, сделать его наиболее удобным и точным. Однако автоматические тесты не могут полностью заменить ручное тестирование. Автоматизация всех испытаний - очень дорогой процесс, и потому автоматическое тестирование является лишь частью ручного тестирования. Наилучший вариант использования автоматических тестов - регрессионное тестирование.

**Задание:** ответить на контрольные вопросы.

Контрольные вопросы.

1. Перечислите основные фазы процесса разработки программного продукта.

2. Назовите методы обоснования истинности формул.

3. В чем заключается отличие статического тестирования от динамического?

4. Назовите этапы тестирования.

5. Каковы основные проблемы тестирования?

6. Каковы требования к идеальному критерию тестирования?

7. Перечислите классы критериев.

8. Перечислите основные уровни тестирования.

9. Каковы цели и задачи модульного тестирования?

10. Какова цель интеграционного тестирования?

11. Какова основная задача системного тестирования?

12. Каковы цели и задачи регрессионного тестирования?