**Практическая работа**

**«Тестирование программных продуктов.** **Выявление первичных и вторичных ошибок. Анализ рисков.»**

ЦЕЛИ РАБОТЫ:

1. Ознакомление с методами тестирования программного продукта.

2. Ознакомиться документацией на разработку программного обеспечения;

3. Сравнить результаты тестирования с требованиями к программным продуктам.

4. Научиться производить анализ рисков программного обеспечения.

5. Научиться выявлять первичные и вторичные ошибки программного обеспечения.

ОБОРУДОВАНИЕ: ПК, C#.

ВРЕМЯ ВЫПОЛНЕНИЯ: 90 минут

КРАТКАЯ ТЕОРИЯ И МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ:

**Тестирование программных продуктов**

***Тестирование*** является завершающим этапом разработки программного продукта. В узком смысле цель тестирования состоит в обнаружении ошибок, цель же отладки – не только в обнаружении, но и в устранении ошибок. Однако ограничиться только отладкой программы, если есть уверенность в том, что все ошибки в ней устранены, нельзя. Цели у отладки и испытания разные.

Полностью отлаженная программа может не обладать определенными потребительскими свойствами и тем самым быть непригодной к использованию по своему назначению.

Не может служить альтернативой испытанию и проверка работоспособности программы на контрольном примере, так как программа, работоспособная в условиях контрольного примера, может оказаться неработоспособной в других условиях применения.

Попытки охватить контрольным примером все предполагаемые условия функционирования сводятся в конечном счете к тем же испытаниям.

В соответствии с ГОСТ 19.004 – 80 под ***испытанием программ*** понимают установление соответствия программы заданным требованиям и программным документам. Это определение построено на предположении, что в техническом задании на разработку программы определены все требования, обеспечение которых гарантирует пригодность программы к использованию по своему назначению.

**Методы тестирования:**

1. Восходящее тестирование – программа собирается и тестируется снизу-вверх.
2. Нисходящее тестирование – программа собирается и тестируется сверху вниз. Изолировано тестируется только головной модуль.
3. Метод большого скачка – каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.
4. Метод сандвича – представляет собой компромисс между восходящим и нисходящим подходами. При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь, в конце концов, где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры.

**Сравнение результатов тестирования с требованиями технического задания**

***Техническое задание*** – это основной исходный документ для разработки программного обеспечения, в котором формулируются основные цели разработки, список принципиальных требований к продукту, определяются сроки и этапы разработки и регламентируется процесс приемно-сдаточных испытаний. Этот документ содержит основные требования заказчика, исходные данные для разработки, указываются назначение продукта, область его применения, стадии разработки различной документации, её состав, сроки исполнения и т. д., а также особые требования, обусловленные спецификой проекта либо условиями его эксплуатации.

Основой технического задания, как документа определяющего характеристики разрабатываемого программного обеспечения и функциональную составляющую программного обеспечения являются: исходные данные и требуемые результаты, которые определяют функции разрабатываемого программного обеспечения; среда (программная и аппаратная), в которой разрабатываемое программное обеспечение будет функционировать, может быть задана, а может выбираться для обеспечения требуемых параметров; возможное взаимодействие с другим программным обеспечением и/или конкретными техническими средствами - также может быть определено, а может выбираться исходя из набора выполняемых функций.

***Спецификация требований программного обеспечения*** – это документ, который содержит законченное описание поведения программы, которую требуется разработать. Включает ряд пользовательских сценариев, которые описывают все варианты взаимодействия между пользователями и программным обеспечением.

Пользовательские сценарии являются средством представления функциональных требований. В дополнение к пользовательским сценариям, спецификация также содержит нефункциональные требования, которые налагают ограничения на дизайн или реализацию (такие как требования производительности, стандарты качества, или проектные ограничения). Состоит из следующих разделов: введение, общее описание, функции системы, требования к данным, требования к внешним интерфейсам, атрибуты качества.

**Анализ рисков**

***Управление рисками*** – это одна из самых динамично развивающихся видов деятельности и большинство крупных компаний имеют в штате специалистов по управлению ими. Существует развитый инструментарий для минимизации потерь от воздействия неблагоприятных факторов как внутренней, так и внешней среды.

Полученные в работе результаты по выявлению, описанию и классификации рисков разработки программного обеспечения являются начальным этапом управления этими рисками. В дальнейшем необходимо оценить риски выбранными измерителями и выбрать управляющие стратегии, после чего оценить их воздействие.

***Риски плохого взаимодействия между заказчиком и исполнителем*** – это риски связанные с отсутствием коммуникации между исполнителем и заказчиком или их представителями. Недостаточное обсуждение задач или архитектуры может негативно сказаться на разрабатываемом программном обеспечении.

***Риски управления проектом*** – это риски, связанные с отсутствием навыков проектного менеджмента у менеджера проекта, а также с отсутствием интереса или мотивации у него. Сама по себе уже хорошо отлаженная система управления рисками может являться эффективным средством для того чтобы определить такого рода риски, так как позволяет идентифицировать проблему и выработать решение.

***Риски, связанные с недостаточной осведомлённостью управляющего проектом о точном состоянии проекта*** – это вид рисков, связанных с отсутствием обратной связи. Он возникает, когда проектный менеджер не выстроил рабочий процесс таким образом, чтобы контролировать ход выполнения проекта на всех его этапах.

***Риски планирования*** – это риски, которые могут быть связан с отсутствием навыков планирования по проекту как менеджером, так и исполнителями, если они готовят информацию о сроках выполнения работ.

***Риски отсутствия системы контроля*** – обусловлены большим количеством аспектов в области проектного менеджмента при разработке программного обеспечения, когда сложно учесть все возможные ситуации.

***Риск появления новых требований*** возникает в процессе разработки программного обеспечения, когда появляются всё новые и новые требования, которые отодвигают сроки и оценку конкретных задач.

***Риск противоречивости в требованиях (декомпозиция спецификации)*** – это риски связанные с выявлением противоречивости в требованиях заказчика на этапе программирования или интеграции проекта.

***Риски неправильно определённых системных требований*** – это риски, когда в самом начале проекта были некорректно сформулированы характеристики целевой системы, для которой разрабатывается программное обеспечение: программное окружение или требования к аппаратной части.

***Риски использования нестабильных технологий*** – это риски, связанные с использованием новых технологий, которые ещё не прошли апробацию в производстве или других проектах.

***Риски, связанные с неспособностью справиться со сложностью проекта*** – иногда проект может быть настолько сложным, что команда попросту может с ним не справиться.

***Риск низкой продуктивности*** обусловлен длительностью реализации проекта. Это в самом начале проекта создаёт большую потерю времени, которую сложно будет наверстать. При этом приходится либо переносить сроки, либо работать в более динамичном режиме на более поздних этапах проекта.

***Риск смены сотрудников***, когда проект покидают ключевые сотрудники, которые максимально владеют информацией.

***Риски хищения исходного кода*** возникают, когда разработчики, уходя из компании, забирают с собой разрабатываемый ими проект и немного модифицировав исходный код, могут продать его или использовать в других проектах, например, у конкурентов.

***Риски нарушение закона об авторском праве*** могут возникнуть при использовании разработчиками без ведома проектного менеджера чужого исходного кода, алгоритма или библиотеки, которые защищены законом об авторском праве, но не приобретены или их использование не согласовано с автором.

**Выявление первичных и вторичных ошибок**

***Под ошибкой*** в широком смысле слова понимается неправильность, погрешность или неумышленное, невольное искажение объекта или процесса. При этом подразумевается, что известно правильное или неискаженное эталонное состояние объекта, к которому относится ошибка.

Считается, что если программа не выполняет того, что пользователь от нее ожидает, то в ней имеется ошибка.

Важной особенностью процесса выявления ошибок в сложных программах является отсутствие полностью определенной правильной программы-эталона, которому должен соответствовать проверяемый текст.

Поэтому нельзя гарантированно утверждать, что возможно написать программу без ошибок.

***Искажения в тексте программ (первичные ошибки)*** являются элементами, подлежащими корректировке. Однако непосредственно наличие ошибки обнаруживаются по ее вторичным проявлениям.

***Искажение выходных результатов исполнения программ (вторичная ошибка)*** вызывает необходимость выполнения ряда операций по локализации устранению первичной ошибки (отладка программ).

На этапе отладки программ выявляются и исправляются много ошибок, но не все.

После отладки в течение некоторого времени интенсивность обнаружения ошибок при самом активном тестировании снижается настолько, что разработчики попадают в зону нечувствительности к программным ошибкам и отказам.

При такой интенсивности отказов программ трудно прогнозировать затраты времени, необходимые для обнаружения очередной ошибки.

Создается представление о полном отсутствии ошибок в программе, о невозможности и бесцельности их поиска. Поэтому усилия на отладку сокращаются, и интенсивность обнаружения ошибок еще больше снижается.

Этой предельно низкой интенсивности обнаружения отказов соответствует наработка на обнаруженную ошибку, при которой прекращается улучшение характеристик программного обеспечения на этапах его отладки и испытаний у заказчика.

Ошибку можно отнести к одному из ниже перечисленных классов:

 системные ошибки;

 ошибки в выборе алгоритма;

 алгоритмические ошибки;

 технологические ошибки;

 программные ошибки.

**Системные ошибки** в большом (сложном) программном обеспечении определяются, прежде всего неполной информацией о реальных процессах, происходящих в источниках и потребителях информации.

На начальных стадиях проектирования программного обеспечения не всегда удается точно сформулировать целевую задачу всей системы и требования к ней. В процессе проектирования целевая функция системы уточняется и выявляются отклонения от уточненных требований, которые могут квалифицироваться как системные ошибки.

Некачественное определение требований к программе приводит к созданию программы, которая будет правильно решать неверно сформулированную задачу. В таких случаях, как правило, требуется полное перепрограммирование.

Признаком того, что создаваемая для заказчика программа может оказаться не соответствующей его истинным потребностям, служит ощущение неясности задачи. Письменная регистрация требований к программе заставляет заказчика собраться с мыслями и дать достаточно точное определение требований. Всякие устные указания являются заведомо ненадежными и часто приводят к взаимному недопониманию.

При автономной и в начале комплексной отладки программного обеспечения доля найденных системных ошибок в нем невелика (примерно 10%), но она существенно возрастает (до 35—40%) на завершающих этапах комплексной отладки.

В процессе эксплуатации преобладающими являются системные ошибки (примерно 80% всех ошибок).

**Ошибки в выборе алгоритма.** Часто плохой выбор алгоритма становится очевидным лишь после его опробования. Поэтому все же следует уделять внимание и время выбору алгоритма, с тем, чтобы впоследствии не приходилось переделывать каждую программу. Во избежание выбора некорректных алгоритмов, необходимо хорошо ознакомиться с литературой по своей специальности.

К алгоритмическим ошибкам следует отнести, прежде всего, ошибки, обусловленные некорректной постановкой функциональных задач, когда в спецификациях не полностью оговорены все условия, необходимые для получения правильного результата.

Эти условия формируются и уточняются в значительной части в процессе тестирования и выявления ошибок в результатах функ-ционирования программ.

Также следует отнести ошибки связей модулей и функциональных групп программ. Их можно квалифицировать как ошибки некорректной постановки задачи.

Алгоритмические ошибки проявляются в неполном учете диапазонов изменения переменных, в неправильной оценке точности используемых и получаемых величин, в неправильном учете связи между различными переменными, в неадекватном представлении формализованных условий решения задачи в спецификациях или схемах, подлежащих программированию и т.д.

Эти обстоятельства являются причиной того, что для исправления каждой алгоритмической ошибки приходится изменять иногда целые ветви программного обеспечения, т.е. пока еще существенно больше операторов, чем при исправлении программных ошибок.

Алгоритмические ошибки значительно труднее поддаются обнаружению методами формализованного автоматического контроля. Вот почему необходимо тщательным образом продумывать алгоритм прежде, чем транслировать его в программу.

Некоторые программисты проверяют алгоритм следующим образом. Через несколько дней после составления алгоритма они повторно обращаются к описанию задачи и составляют алгоритм заново. Затем сличают оба варианта. Такой шаг на первый взгляд может показаться пустой тратой времени, однако всякая ошибка на уровне алгоритма может в дальнейшем обернуться катастрофой и повлечь основательный пересмотр программы.

**Технологические ошибки**— это ошибки документации и фиксирования программ в памяти ЭВМ. Они составляют 5—10 % от общего числа ошибок, обнаруживаемых при отладке. Большинство технологических ошибок

выявляются автоматически формализованными методами (например, транслятором).

**Программные ошибки**. Языки программирования - это искусственные языки, созданные человеком для описания алгоритмов. Все предложения таких языков строятся по строгим синтаксическим правилам, обеспечивающим однозначное их понимание, что позволяет поручать расшифровку алгоритма ЭВМ, построенного по правилам семантики.

***Синтаксис*** - это набор правил построения из символов алфавита специальных конструкций, с помощью которых можно составлять различные алгоритмы (программы). Эти правила требуют их неукоснительного соблюдения. В противном случае будет нарушен основной принцип - четкая и строгая однозначность в понимании алгоритма.

***Семантика языка*** — это система правил истолкования построений конструкций. Правила семантики конструкций обычно вполне естественны и понятны, но в некоторых случаях их надо специально оговаривать, комментировать.

Таким образом, программы, позволяющие однозначно производить процесс переработки данных, составляются с помощью соединения символов из алфавита в предложения в соответствии с синтаксическими правилами, определяющими язык, с учетом правил семантики.

**Выделяют синтаксические и семантические ошибки.**

***Под синтаксическими ошибками*** понимается нарушение правил записи программ на данном языке программирования. Они выявляются самой машиной, точнее транслятором, вовремя перевода записи алгоритма на язык машины. Исправление их осуществляется просто — достаточно сравнить формат исправляемой конструкции с синтаксисом в справочнике и исправить его.

***Семантические (смысловые) ошибки*** — это применение операторов, которые не дают нужного эффекта (например, а—вместо, а+в), ошибка в структуре алгоритма, в логической взаимосвязи его частей, в применении

алгоритма к тем данным, к которым он неприменим и т.д. Правила семантики не формализуемы. Поэтому поиск и устранение семантической ошибки и составляет основу отладки.

Каждая программная ошибка влечет за собой необходимость изменения команд существенно меньше, чем при алгоритмических и системных ошибках.

На этапах комплексной отладки программного обеспечения и эксплуатации удельный вес программных ошибок падает и составляет примерно 15 и 30 % соответственно от общего количества ошибок, выявляемых в единицу времени.

***ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ И ФОРМА ОТЧЕТНОСТИ:***

**Задание 1. Разработать план тестирования программного продукта (тест план).** Смотри пример тест – плана.

**Задание 2. Сравнение результатов тестирования с требованиями технического задания и спецификации.**

1. Составьте техническое задание, которое будет содержать необходимые требования к программному продукту.

2. Составьте спецификацию на разработку программного продукта.

3. Сравните результаты тестирования из предыдущего задания с требованиями технического задания.

4. Сравните результаты тестирования из предыдущего задания с требованиями спецификации.

**Задание 3. Определение рисков при разработке программного продукта.**

Заполните таблицу.

**Риски разработки программного обеспечения**

|  |  |  |  |
| --- | --- | --- | --- |
| Риск | Способ устранения | Последствия, связанные с не устранением риска | Меры минимизации риска |
|  |  |  |  |

**Задание 4. Определение видов ошибок в программном продукте.**

Заполните таблицу.

**Категории тяжести ошибки в программном обеспечении**

|  |  |  |
| --- | --- | --- |
| Номер категории ошибки | Наименование категории тяжести ошибки | Описание последствий проявления ошибки |
|  |  |  |

Разделите ошибки в программном продукте на первичные и на вторичные. Заполните таблицу.

|  |  |  |
| --- | --- | --- |
| Описание ошибки | Вид ошибки (первичные или вторичные) | Методы устранения |
|  |  |  |