Лекция

Тема: Ручное и автоматизированное тестирование

«Отчего у нас никогда нет времени сделать что-либо хорошо, но всегда находится время на переделку?»

Бытует мнение, что первая программная ошибка была обнаружена на заре развития ЭВМ, когда в Массачусетском технологическом институте окончилась неудачей попытка запуска машины Whirlwind I («Вихрь I»). Неистовая проверка монтажа, соединений и оборудования не выявила никаких неисправностей. Наконец, уже отчаявшись, решили проверить программу, представляющую собой маленькую полоску бумажной ленты. И ошибка была обнаружена именно в ней – в этом программистском ящике Пандоры[1], из которого на будущие поколения программистов обрушились беды, связанные с ошибками программ.

Задача любого тестировщика заключается в нахождении наибольшего количества ошибок, поэтому он должен хорошо знать наиболее часто допускаемые ошибки и уметь находить их за минимально короткий период времени. Остальные ошибки, которые не являются типовыми, обнаруживаются только тщательно созданными наборами тестов. Однако, из этого не следует, что для типовых ошибок не нужно составлять тесты.

Далее будет дана классификация ошибок, что поможет сосредоточить наши усилия в правильном направлении.

1. Классификация ошибок

Для классификации ошибок мы должны определить термин «ошибка».

Ошибка – это расхождение между вычисленным, наблюдаемым и истинным, заданным или теоретически правильным значением [7].

Такое определение понятия «ошибка» не является универсальным, так как оно больше подходит для понятия «программная ошибка». В технологии программирования существуют не только программные ошибки, но и ошибки, связанные с созданием программного продукта, например, ошибки в документации программы. Отличие программы и программного продукта достаточно четко определены в [8]. Но нас пока будут интересовать программные ошибки.

Итак, *по времени появления* ошибки можно разделить на три вида:

- **структурные ошибки набора**;
- **ошибки компиляции**;
- **ошибки периода выполнения**.

Структурные ошибки возникают непосредственно при наборе программы. Что это за ошибки? Если кто-то работал в среде разработки Microsoft Visual Basic, то он знает, что если набрать оператор **If**, затем сравнение и нажать на клавишу **Enter**, не набрав слова **Then**, то Visual Basic укажет, что возникла ошибка компиляции. Это не совсем верно, так как компиляция в Visual Basic происходит только непосредственно

[1] Пандора – в древнегреческой мифологии девушка, созданная из земли и воды богом огня и кузнечного ремесла Гефестом. Она получила от верховного бога Зевса ящик со всеми человеческими несчастьями, которые случайно выпустила, приоткрыв из любопытства крышку; отсюда «ящик Пандоры» – источник всяческих бедствий.

при выполнении команды программы. В данном случае мы имеем дело именно со структурной ошибкой набора.

Данный тип ошибок определяется либо при наборе программы (самой IDE (**I**ntegrated **D**evelopment **E**nvironment) – интегрированной средой разработки) или при ее компиляции, если среда не разделяет первые два типа ошибок.

К данному типу ошибок относятся такие как: несоответствие числа открывающих скобок числу закрывающих, отсутствие парного оператора (например, **try** без **catch**), неправильное употребление синтаксических знаков и т. п.

Во многих средах разработки программного обеспечения данный тип ошибок объединяется со следующим типом, так как раннее определение ошибок вызывает некоторое неудобство при наборе программ (скажем, я задумал что-то написать, а потом вспомнил, что в начале пропустил оператор, тогда среда разработки может выдать мне ошибку при попытке перейти в другую строку).

Еще раз нужно отметить, что данный тип ошибок достаточно уникален и выделяется в отдельный тип только некоторыми средами разработки программного обеспечения.

Ошибки компиляции возникают из-за ошибок в тексте кода. Они включают ошибки в синтаксисе, неверное использование конструкций языка (оператор **else** в операторе **for** и т. п.), использование несуществующих объектов или свойств, методов у объектов.

Среда разработки (компилятор) обнаружит эти ошибки при общей компиляции приложения и сообщит о **последствиях** этих ошибок. Необходимо подчеркнуть слово «последствия» – это очень важно. Дело в том, что часто, говоря об ошибках, мы не разделяем проявление ошибки и саму ошибку, хотя это и не одно и то же. Например, ошибка «неопределенный класс» не означает, что класс не определен. Он может быть неподключенным, так как не подключен пакет классов.

Ошибки периода выполнения возникают, когда программа выполняется и компилятор (или операционная система, виртуальная машина) обнаруживает, что оператор делает попытку выполнить недопустимое или невозможное действие. Например, деление на ноль. Предположим, имеется такое выражение:

```
ratio = firstValue / sum.
```

Если переменная sum содержит ноль, то деление – недопустимая операция, хотя сам оператор синтаксически правилен. Прежде, чем программа обнаружит эту ошибку, ее необходимо запустить на выполнение.

Хотя данный тип ошибок называется «ошибками периода выполнения», это не означает, что ошибки находятся только после запуска программы. Вы можете выполнять программу в уме и обнаружить ошибки данного типа, однако, понятно, что это крайне неэффективно.

Если проанализировать все типы ошибок согласно первой классификации, то можно прийти к заключению, что при тестировании приходится иметь дело с ошибками периода выполнения, так как первые два типа ошибок определяются на этапе кодирования.

В теоретической информатике программные ошибки классифицируют *по степени нарушения логики на*:

- **синтаксические**;
- **семантические**;
- **прагматические**.

Синтаксические ошибки заключаются в нарушении правописания или пунктуации в записи выражений, операторов и т. п., т. е. в нарушении грамматических правил языка. В качестве примеров синтаксических ошибок можно назвать:

- пропуск необходимого знака пунктуации;
- несогласованность скобок;
- пропуск нужных скобок;
- неверное написание зарезервированных слов;
- отсутствие описания массива.

Все ошибки данного типа обнаруживаются компилятором.

Семантические ошибки заключаются в нарушении порядка операторов, параметров функций и употреблении выражений. Например, параметры у функции **add** (на языке Java) в следующем выражении указаны в неправильном порядке:

```
GregorianCalendar.add(1, Calendar.MONTH).
```

Параметр, указывающий изменяемое поле (в примере – месяц), должен идти первым. Семантические ошибки также обнаруживаются компилятором.

Надо отметить, что некоторые исследователи относят семантические ошибки к следующей группе ошибок.

Прагматические ошибки (или логические) заключаются в неправильной логике алгоритма, нарушении смысла вычислений и т. п. Они являются самыми сложными и крайне трудно обнаруживаются. Компилятор может выявить только следствие прагматической ошибки (см. выше пример с делением на ноль, компилятор обнаружит деление на ноль, но когда и почему переменная sum стала равна нулю – должен найти программист).

Таким образом, после рассмотрения двух классификаций ошибок можно прийти к выводу, что на этапе тестирования ищутся прагматические ошибки периода выполнения, так как остальные выявляются в процессе программирования.

На этом можно было бы закончить рассмотрение классификаций, но с течением времени накапливался опыт обнаружения ошибок и сами ошибки, некоторые из которых образуют характерные группы, которые могут тоже служить характерной классификацией.

Ошибка адресации – ошибка, состоящая в неправильной адресации данных (например, выход за пределы участка памяти).

Ошибка ввода-вывода – ошибка, возникающая в процессе обмена данными между устройствами памяти, внешними устройствами.

Ошибка вычисления – ошибка, возникающая при выполнении арифметических операций (например, разнотипные данные, деление на нуль и др.).

Ошибка интерфейса – программная ошибка, вызванная несовпадением характеристик фактических и формальных параметров (как правило, семантическая ошибка периода компиляции, но может быть и логической ошибкой периода выполнения).

Ошибка обращения к данным – ошибка, возникающая при обращении программы к данным (например, выход индекса за пределы массива, не инициализированные значения переменных и др.).

Ошибка описания данных – ошибка, допущенная в ходе описания данных.

2. Первичное выявление ошибок

В течение многих лет большинство программистов убеждено в том, что программы пишутся исключительно для выполнения их на машине и не

предназначены для чтения человеком, а единственным способом тестирования программы является ее исполнение на ЭВМ. Это мнение стало изменяться в начале 70-х годов в значительной степени благодаря книге Вейнберга «Психология программирования для ЭВМ» [9]. Вейнберг показал, что программы должны быть удобочитаемыми и что их просмотр должен быть эффективным процессом обнаружения ошибок.

По этой причине, прежде чем перейти к обсуждению традиционных методов тестирования, основанных на применении ЭВМ, рассмотрим процесс тестирования без применения ЭВМ («ручное тестирование»), являющийся по сути первичным обнаружением ошибок. Эксперименты показали, что методы ручного тестирования достаточно эффективны с точки зрения нахождения ошибок, так что один или несколько из них должны использоваться в каждом программном проекте. Описанные здесь методы предназначены для периода разработки, когда программа закодирована, но тестирование на ЭВМ еще не началось. Аналогичные методы могут быть получены и применены на более ранних этапах процесса создания программ (т. е. в конце каждого этапа проектирования).

Следует заметить, что из-за неформальной природы методов ручного тестирования (неформальной с точки зрения других, более формальных методов, таких, как математическое доказательство корректности программ) первой реакцией часто является скептицизм, ощущение того, что простые и неформальные методы не могут быть полезными. Однако их использование показало, что они не «уводят в сторону». Скорее эти методы способствуют существенному увеличению производительности и повышению надежности программы. Во-первых, они обычно позволяют раньше обнаружить ошибки, уменьшить стоимость исправления последних и увеличить вероятность того, что корректировка произведена правильно. Вовторых, психология программистов, по-видимому, изменяется, когда начинается тестирование на ЭВМ. Возрастает внутреннее напряжение и появляется тенденция «исправлять ошибки так быстро, как только это возможно». В результате программисты допускают больше промахов при корректировке ошибок, уже найденных во время тестирования на ЭВМ, чем при корректировке ошибок, найденных на более ранних этапах.

Кроме того, скептицизм связан с тем, что это «первобытный метод». Да, сейчас стоимость машинного времени очень низка, а стоимость труда программиста, тестировщика высока и ряд руководителей пойдут на все, чтобы сократить расходы. Однако, есть другая сторона ручного тестирования – при тестировании за компьютером причины ошибок выявляются только в программе, а самая глубокая их причина – мышление программиста, как правило, не претерпевает изменений, при ручном же тестировании, программист глубоко анализирует свой код, попутно выявляя возможные пути его оптимизации, и изменяет собственный стиль мышления, повышая квалификацию. Таким образом, можно прийти к выводу, что ручное тестирование можно и нужно проводить на первичном этапе, особенно, если нет прессинга времени и бюджета.

3. Инспекции и сквозные просмотры

Инспекции исходного текста и сквозные просмотры являются основными методами ручного тестирования. Так как эти два метода имеют много общего, они рассматриваются здесь совместно.

Инспекции и сквозные просмотры включают в себя чтение или визуальную проверку программы группой лиц. Эти методы развиты из идей Вейнберга [9]. Оба

метода предполагают некоторую подготовительную работу. Завершающим этапом является «обмен мнениями» – собрание, проводимое участниками проверки. Цель такого собрания – нахождение ошибок, но не их устранение (т. е. тестирование, а не отладка).

Инспекции и сквозные просмотры широко практикуются в настоящее время, но причины их успеха до сих пор еще недостаточно выяснены. Заметим, что данный процесс выполняется группой лиц (оптимально три-четыре человека), лишь один из которых является автором программы. Следовательно, программа, по существу, тестируется не автором, а другими людьми, которые руководствуются изложенными ранее принципами (в разделе 1), обычно не эффективными при тестировании собственной программы. Фактически «инспекция» и «сквозной просмотр» – просто новые названия старого метода «проверки за столом» (состоящего в том, что программист просматривает свою программу перед ее тестированием), однако они гораздо более эффективны опять-таки по той же причине: в процессе участвует не только автор программы, но и другие лица. Результатом использования этих методов является, обычно, точное определение природы ошибок. Кроме того, с помощью данных методов обнаруживают группы ошибок, что позволяет в дальнейшем корректировать сразу несколько ошибок. С другой стороны, при тестировании на ЭВМ обычно выявляют только симптомы ошибок (например, программа не закончилась или напечатала бессмысленный результат), а сами они определяются поодиночке.

Ранее, более двух десятков лет, проводились широкие эксперименты по применению этих методов, которые показали, что с их помощью для типичных программ можно находить от 30 до 70 % ошибок логического проектирования и кодирования. (Однако эти методы не эффективны при определении ошибок проектирования «высокого уровня», например, сделанных в процессе анализа требований.) Так, было экспериментально установлено, что при проведении инспекций и сквозных просмотров определяются в среднем 38 % общего числа ошибок в учебных программах [12]. При использовании инспекций исходного текста в фирме IBM эффективность обнаружения ошибок составляла 80 % [13] (в данном случае имеется в виду не 80 % общего числа ошибок, поскольку, как отмечалось ранее, общее число ошибок в программе никогда не известно, а 80 % всех ошибок, найденных к моменту окончания процесса тестирования).

Конечно, можно критиковать эту статистику в предположении, что ручные методы тестирования позволяют находить только «легкие» ошибки (те, которые можно просто найти при тестировании на ЭВМ), а трудные, незаметные или необычные ошибки можно обнаружить только при тестировании на машине. Однако проведенное исследование показало, что подобная критика является необоснованной [14]. Кроме того, можно было бы утверждать, что ручное тестирование «морально устарело», но если обратить внимание на список типовых ошибок, то они до сих пор остались прежними и увеличит ли скорость тестирования ЭВМ не всегда очевидно. Но то, что эти методы стали совсем непопулярными – это факт. Бесспорно, что каждый метод хорош для своих типов ошибок и сочетание методов ручного тестирования и тестирования с применением ЭВМ для конкретной команды разработчиков представляется наиболее эффективным подходом; эффективность обнаружения ошибок уменьшится, если тот или иной из этих подходов не будет использован.

Наконец, хотя методы ручного тестирования весьма важны при тестировании новых программ, они представляют не меньшую ценность при тестировании

модифицированных программ. Опыт показал, что в случае модификации существующих программ вносится большее число ошибок (измеряемое числом ошибок на вновь написанные операторы), чем при написании новой программы. Следовательно, модифицированные программы также должны быть подвергнуты тестированию с применением данных методов.

1. Инспекции исходного текста

Инспекции исходного текста представляют собой набор процедур и приемов обнаружения ошибок при изучении (чтении) текста группой специалистов [15]. При рассмотрении инспекций исходного текста внимание будет сосредоточено в основном на методах, процедурах, формах выполнения и т. д.

Инспектирующая группа включает обычно четыре человека, один из которых выполняет функции председателя. Председатель должен быть компетентным программистом, но не автором программы; он не должен быть знаком с ее деталями. В обязанности председателя входят подготовка материалов для заседаний инспектирующей группы и составление графика их проведения, ведение заседаний, регистрация всех найденных ошибок и принятие мер по их последующему исправлению. Председателя можно сравнить с инженером отдела технического контроля. Членами группы являются автор программы, проектировщик (если он не программист) и специалист по тестированию.

Общая процедура заключается в следующем. Председатель заранее (например, за несколько дней) раздает листинг программы и проектную спецификацию остальным членам группы. Они знакомятся с материалами до заседания. Инспекционное заседание разбивается на две части:

1. Программиста просят рассказать о логике работы программы. Во время беседы возникают вопросы, преследующие цель обнаружения ошибки. Практика показала, что даже только чтение своей программы слушателям представляется эффективным методом обнаружения ошибок и многие ошибки находит сам программист, а не другие чле-

ны группы. Этот феномен известен давно и часто его применяют для решения проблем. Когда решение неочевидно, то объяснение проблемы другому человеку заставляет разработчика «разложить все по полочкам» и решение «само приходит» к разработчику.

2. Программа анализируется по списку вопросов для выявления исторически сложившихся общих ошибок программирования.

Председатель является ответственным за обеспечение результативности дискуссии. Ее участники должны сосредоточить свое внимание на нахождении ошибок, а не на их корректировке. (Корректировка ошибок выполняется программистом после инспекционного заседания.)

По окончании заседания программисту передается список найденных ошибок. Если список включает много ошибок или если эти ошибки требуют внесения значительных изменений, председателем может быть принято решение о проведении после корректировки повторной инспекции программы. Список анализируется и ошибки распределяются по категориям, что позволяет совершенствовать его с целью повышения эффективности будущих инспекций. Можно даже вести учет типов ошибок, на основании которого следует проводить дополнительную стажировку программиста в слабых областях.

В большинстве примеров описания процесса инспектирования утверждается, что во время инспекционного заседания ошибки не должны корректироваться. Однако

существует и другая точка зрения [16]: «Вместо того, чтобы сначала сосредоточиться на основных проблемах проектирования, необходимо решить второстепенные вопросы. Два или три человека, включая разработчика программы, должны внести очевидные исправления в проект с тем, чтобы впоследствии решить главные задачи. Однако обсуждение второстепенных вопросов сконцентрирует внимание группы на частной области проектирования. Во время обсуждения наилучшего способа внесения изменений в проект кто-либо из членов группы может заметить еще одну проблему. Теперь группе придется рассматривать две проблемы по отношению к одним и тем же аспектам проектирования, объяснения будут полными и быстрыми. В течение нескольких минут целая область проекта может быть полностью исследована и любые проблемы станут очевидными... Как упоминалось выше, многие важные проблемы, возникавшие во время обзоров блок-схем, были решены в результате многократных безуспешных попыток решить вопросы, которые на первый взгляд казались тривиальными».

Время и место проведения инспекции должны быть спланированы так, чтобы избежать любых прерываний инспекционного заседания. Его оптимальная продолжительность, по-видимому, лежит в пределах от 90 до 120 мин. Так как это заседание является экспериментом, требующим умственного напряжения, увеличение его продолжительности ведет к снижению продуктивности. Большинство инспекций происходит при скорости, равной приблизительно 150 строк в час. При этом подразумевается, что большие программы должны рассматриваться за несколько инспекций, каждая из которых может быть связана с одним или несколькими модулями или подпрограммами.

Для того чтобы инспекция была эффективной, должны быть установлены соответствующие отношения. Если программист воспринимает инспекцию как акт, направленный лично против него, и, следовательно, занимает оборонительную позицию, процесс инспектирования не будет эффективным. Программист должен подходить к нему с менее эгоистических позиций [9]; он должен рассматривать инспекцию в позитивном и конструктивном свете: объективно инспекция является процессом нахождения ошибок в программе и таким образом улучшает качество его работы. По этой причине, как правило, рекомендуется результаты инспекции считать конфиденциальными материалами, доступными только участникам заседания. В частности, использование результатов инспекции руководством может нанести ущерб целям этого процесса.

Процесс инспектирования в дополнение к своему основному назначению, заключающемуся в нахождении ошибок, выполняет еще ряд полезных функций. Кроме того, что результаты инспекции позволяют программисту увидеть сделанные им ошибки и способствуют его обучению на собственных ошибках, он обычно получает возможность оценить свой стиль программирования и выбор алгоритмов и методов тестирования. Остальные участники также приобретают опыт, рассматривая ошибки и стиль программирования других программистов.

Наконец, инспекция является способом раннего выявления наиболее склонных к ошибкам частей программы, позволяющим сконцентрировать внимание на этих частях в процессе выполнения тестирования на ЭВМ (один из принципов тестирования [1]).

2. Сквозные просмотры

Сквозной просмотр, как и инспекция, представляет собой набор процедур и способов обнаружения ошибок, осуществляемых группой лиц, просматривающих

текст программы. Такой просмотр имеет много общего с процессом инспектирования, но их процедуры несколько отличаются и, кроме того, здесь используются другие методы обнаружения ошибок.

Подобно инспекции, сквозной просмотр проводится как непрерывное заседание, продолжающееся один или два часа. Группа по выполнению сквозного просмотра состоит из 3–5 человек. В нее входят председатель, функции которого подобны функциям председателя в группе инспектирования, секретарь, который записывает все найденные ошибки, и специалист по тестированию. Мнения о том, кто должен быть четвертым и пятым членами группы, расходятся. Конечно, одним из них должен быть программист. Относительно пятого участника имеются следующие предположения: 1) высококвалифицированный программист; 2) эксперт по языку программирования; 3) начинающий (на точку зрения которого не влияет предыдущий опыт); 4) человек, который будет, в конечном счете, эксплуатировать программу; 5) участник какого-нибудь другого проекта; 6) кто-либо из той же группы программистов, что и автор программы.

Начальная процедура при сквозном просмотре такая же, как и при инспекции: участникам заранее, за несколько дней до заседания, раздаются материалы, позволяющие им ознакомиться с программой. Однако эта процедура отличается от процедуры инспекционного заседания. Вместо того, чтобы просто читать текст программы или использовать список ошибок, участники заседания «выполняют роль вычислительной машины». Лицо, назначенное тестирующим, предлагает собравшимся небольшое число написанных на бумаге тестов, представляющих собой наборы входных данных (и ожидаемых выходных данных) для программы или модуля. Во время заседания каждый тест мысленно выполняется. Это означает, что тестовые данные подвергаются обработке в соответствии с логикой программы. Состояние программы (т. е. значения переменных) отслеживается на бумаге или доске.

Конечно, число тестов должно быть небольшим и они должны быть простыми по своей природе, потому что скорость выполнения программы человеком на много порядков меньше, чем у машины. Следовательно, тесты сами по себе не играют критической роли, скорее они служат средством для первоначального понимания программы и основой для вопросов программисту о логике проектирования и принятых допущениях. В большинстве сквозных просмотров при выполнении самих тестов находят меньше ошибок, чем при опросе программиста.

Как и при инспекции, мнение участников является решающим фактором. Замечания должны быть адресованы программе, а не программисту. Другими словами, ошибки не рассматриваются как слабость человека, который их совершил. Они свидетельствуют о сложности процесса создания программ и являются результатом все еще примитивной природы существующих методов программирования.

Сквозные просмотры должны протекать так же, как и описанный ранее процесс инспектирования. Побочные эффекты, получаемые во время выполнения этого процесса (установление склонных к ошибкам частей программы и обучение на основе анализа ошибок, стиля и методов) характерны и для процессасквозныхпросмотров.

3. Проверка за столом

Третьим методом ручного обнаружения ошибок является применявшаяся ранее других методов «проверка за столом». Проверка за столом может рассматриваться как проверка исходного текста или сквозные просмотры, осуществляемые одним

человеком, который читает текст программы, проверяет его по списку ошибок и (или) пропускает через программу тестовые данные.

Большей частью проверка за столом является относительно непродуктивной. Это объясняется прежде всего тем, что такая проверка представляет собой полностью неупорядоченный процесс. Вторая, более важная причина заключается в том, что проверка за столом противопоставляется одному из принципов тестирования [1], согласно которому программист обычно неэффективно тестирует собственные программы. Следовательно, проверка за столом наилучшим образом может быть выполнена человеком, не являющимся автором программы (например, два программиста могут обмениваться программами вместо того, чтобы проверять за столом свои собственные программы), но даже в этом случае такая проверка менее эффективна, чем сквозные просмотры или инспекции. Данная причина является главной для образования группы при сквозных просмотрах или инспекциях исходного текста. Заседание группы благоприятствует созданию атмосферы здоровой конкуренции: участники хотят показать себя с лучшей стороны при нахождении ошибок. При проверке за столом этот, безусловно, ценный эффект отсутствует. Короче говоря, проверка за столом, конечно, полезна, но она гораздо менее эффективна, чем инспекция исходного текста или сквозной просмотр.

4. Список вопросов для выявления ошибок при инспекции

Важной частью процесса инспектирования является проверка программы на наличие общих ошибок с помощью списка вопросов для выявления ошибок. Концентрация внимания в предлагаемом списке (как, например, в работе [17]) на рассмотрении стиля, а не ошибок (вопросы типа «Являются ли комментарии точными и информативными?» и «Располагаются ли символы THEN/ELSE и DO/END по одной вертикали друг под другом?») представляется неудачной, так же как и наличие неопределенности в списке, уменьшающее его полезность (вопросы типа «Соответствует ли текст программы требованиям, выдвигаемым при проектировании?»). Список, приведенный в данном разделе, был составлен различными авторами. За основу взят список Майерса [18] и дополнен автором после многолетнего изучения ошибок программного обеспечения, разработанного как лично, так и другими специалистами, а также учебных программ. В значительной мере он является независимым от языка; это означает, что большинство ошибок встречается в любом языке программирования. Любой специалист может дополнить этот список вопросами, позволяющими выявить ошибки, специфичные для того языка программирования, который он использует, и обнаруженные им в результате выполнения процесса инспектирования.

1. Ошибки обращения к данным

Сводный список вопросов таков:

1. Используются ли переменные с неустановленными значениями?

Наличие переменных с неустановленными значениями – наиболее часто встречающаяся программная ошибка, она возникает при различных обстоятельствах. Для каждого обращения к единице данных (например, к переменной, элементу массива, полю в структуре, атрибуту в классе) попытайтесь неформально «доказать», что ей присвоено значение в проверяемой точке.

2. Лежат ли индексы вне заданных границ?

Не выходит ли значение каждого из индексов за границы, определенные для соответствующего измерения при всех обращениях к массиву, вектору, списку и т. п.?

3. Есть ли нецелые индексы?

Принимает ли каждый индекс целые значения при всех обращениях к массиву, вектору, списку? Нецелые индексы не обязательно являются ошибкой для всех языков программирования, но представляют практическую опасность.

4. Есть ли «подвешенные» обращения?

Создан ли объект (выделена ли память) для всех обращений с помощью указателей или переменных-ссылок на объект (или память)? Наличие, переменных-ссылок представляет собой ошибку типа «подвешенного обращения». Она возникает в ситуациях, когда время жизни указателя больше, чем время жизни объекта/памяти, к которому/ой производится обращение. Например, к такому результату приводит ситуация, когда указатель задает локальную переменную в теле метода, значение указателя присваивается выходному параметру или глобальной переменной, метод завершается (освобождая адресуемую память), а программа затем пытается использовать значение указателя. Как и при поиске ошибок первых трех типов, попытайтесь неформально доказать, что для каждого обращения, использующего переменную-указатель, адресуемая память/объект существует.

Этот тип ошибок характерен для языка Си или C++, где широко используются ссылки и указатели. Язык Java в этом отношении более развит, например, при потере всех ссылок на объект, объект переходит в «кучу мусора», где автоматически освобождается память из-под объекта (объект удаляется) специальным сборщиком мусора. Последние изменения в языке C++, выполненные командой разработчиков Microsoft, которые преобразовали этот язык в C#, реализуют похожий механизм.

5. Соответствуют ли друг другу определения структуры и ее использование в различных методах?

Если к структуре данных обращаются из нескольких методов или процедур, то определена ли эта структура одинаково в каждой процедуре и используется ли она корректным способом?

6. Превышены ли границы строки?

Не превышены ли границы строки при индексации в ней? Существуют ли какие-нибудь другие ошибки в операциях с индексацией или при обращении к массивам по индексу?

2. Ошибки описания данных

Сводный список вопросов таков:

1. Все ли переменные описаны?

Все ли переменные описаны явно? Отсутствие явного описания не обязательно является ошибкой (например, Visual Basic допускает отсутствие описания), но служит потенциальным источником беспокойства. Так, если в подпрограмме на Visual Basic используется элемент массива и отсутствует его описание (например, в операторе DIM), то обращение к массиву может вызвать ошибку (например, X = A(12)), так как по умолчанию, массив определен только на 10 элементов. Если отсутствует явное описание переменной во внутренней процедуре или блоке, следует ли понимать это так, что описание данной переменной совпадает с описанием во внешнем блоке? При разработке больших программных изделий неявное описание данных (описание данных по умолчанию) зачастую запрещают методически (если это не запрещено языком), чтобы упростить поиск ошибок при комплексной отладке.

2. Правильно ли инициализированы объекты, массивы и строки?

Если начальные значения присваиваются переменным в операторах описания, то правильно ли инициализируются эти значения? Правильно ли создаются объекты, используется ли соответствующий конструктор?

3. Понятны ли имена переменных?

Наличие переменных с бессмысленными именами (например, i и j) не является ошибкой, но является объектом пристального внимания. Классически i и j являются цикловыми переменными, а вот названий типа t125 следует избегать, так как возможна путаница имен.

4. Нельзя ли обойтись без переменных со сходными именами?

Есть ли переменные со сходными именами (например, user и users)? Наличие сходных имен не обязательно является ошибкой, но служит признаком того, что имена могут быть перепутаны гденибудь внутри программы.

5. Корректно ли произведено описание класса?

Правильно ли происходит описание атрибутов и методов класса? Имеются ли методы или атрибуты, которые по смыслу не подходят к данному классу? Не является ли класс громоздким? Наличие положительных ответов на эти вопросы указывает на возможные ошибки в анализе и проектировании системы.

3. Ошибки вычислений

Сводный список вопросов таков:

1. Производятся ли вычисления с использованием данных разного типа?

Существуют ли вычисления, использующие данные разного типа? Например, сложение переменной с плавающей точкой и целой переменной. Такие случаи не обязательно являются ошибочными, но они должны быть тщательно проверены для обеспечения гарантии того, что правила преобразования, принятые в языке, понятны. Это важно как для языков с сильной типизацией (например, Ada, Java), так и для языков со слабой типизацией (например, C++, хотя он тяготеет к сильной типизации). Например, для языка Java код

```
byte a, b, c;
… c = a + b;
```

может вызвать ошибку, так как операция «сложение» преобразует данные к типу int, и результат может превысить максимально возможное значение для типа byte. Таким образом, важным для вычислений с использованием различных типов данных является явное или неявное преобразование типов. Ошибки, связанные с использованием данных разных типов являются одними из самых распространенных.

2. Производятся ли вычисления неарифметических переменных?

3. Возможно ли переполнение или потеря промежуточного результата при вычислении?

Это означает, что конечный результат может казаться правильным, но промежуточный результат может быть слишком большим или слишком малым для машинного представления данных. Ошибки могут возникнуть даже если существует преобразование типов данных.

4. Есть ли деление на ноль?

Классическая ошибка. Требует проверки всех делителей на неравенство нулю. Следствием данной ошибки является либо сообщение «деление на ноль», либо «переполнение», если делитель очень близок к нулю, а результат не может быть сохранен в типе частного (превышает его).

5. Существуют ли неточности при работе с двоичными числами?

6. Не выходит ли значение переменной за пределы установленного диапазона?

Может ли значение переменной выходить за пределы установленного для нее логического диапазона? Например, для операторов, присваивающих значение переменной probability (вероятность), может быть произведена проверка, будет ли полученное значение всегда положительным и не превышающим единицу. Другие диапазоны могут зависеть от области решаемых задач.

7. Правильно ли осуществляется деление целых чисел?

Встречается ли неверное использование целой арифметики, особенно деления? Например, если i – целая величина, то выражение `2*i/2 = i` зависит от того, является значение i четным или нечетным, и от того, какое действие – умножение или деление – выполняется первым.

4. Ошибки при сравнениях

Сводный список вопросов таков:

1. Сравниваются ли величины несовместимых типов? Например, число со строкой?
2. Сравниваются ли величины различных типов?

Например, переменная типа int с переменной типа long? Каждый язык ведет себя в этих случаях по-своему, проверьте это по его описанию. Как выполняются преобразования типов в этих случаях?

3. Корректны ли отношения сравнения?

Иногда возникает путаница понятий «наибольший», «наименьший», «больше чем», «меньше чем».

4. Корректны ли булевские выражения?

Если выражения очень сложные, имеет смысл преобразовать их или проверять обратное утверждение.

5. Понятен ли порядок следования операторов?

Верны ли предположения о порядке оценки и следовании операторов для выражений, содержащих более одного булевского оператора? Иными словами, если задано выражение `(A == 2) && (B == 2) || (C == 3)`, понятно ли, какая из операций выполняется первой: И или ИЛИ?

6. Понятна ли процедура разбора компилятором булевских выражений?

Влияет ли на результат выполнения программы способ, которым конкретный компилятор выполняет булевские выражения? Например, оператор

```
if ((x != 0) && ((y/x) > z))
```

является приемлемым для Java (т. е. компилятор заканчивает проверку, как только одно из выражений в операции И окажется ложным), но это выражение может привести к делению на ноль при использовании компиляторов других языков.

5. Ошибки в передачах управления

Сводный список вопросов таков:

1. Может ли значение индекса в переключателе превысить число переходов? Например, значение переключателя для оператора select case.
2. Будет ли завершен каждый цикл?

Будет ли каждый цикл, в конце концов, завершен? Придумайте неформальное доказательство или аргументы, подтверждающие их завершение. Хотя иногда бесконечные циклы не являются ошибкой, но лучше их избегать.

3. Будет ли завершена программа? Будет ли программа, метод, модуль или подпрограмма в конечном счете завершена?

4. Существует ли какой-нибудь цикл, который не выполняется из-за входных условий?

Возможно ли, что из-за входных условий цикл никогда не сможет выполняться? Если это так, то является ли это оплошностью?

5. Есть ли ошибки отклонения числа итераций от нормы?

Существуют ли какие-нибудь ошибки «отклонения от нормы» (например, слишком большое или слишком малое число итераций)?

6. Ошибки интерфейса

Сводный список вопросов таков:

1. Равно ли число входных параметров числу аргументов?

Равно ли число параметров, получаемых рассматриваемым методом, числу аргументов, ему передаваемых каждым вызывающим методом? Правилен ли порядок их следования? Первый тип ошибок может обнаруживаться компилятором (но не для каждого языка), а вот правильность следования (особенно, если параметры одинакового типа) является важным моментом.

2. Соответствуют ли единицы измерения параметров и аргументов?

Например, нет ли случаев, когда значение параметра выражено в градусах, а аргумента – в радианах? Или ошибки связанные с размерностью параметра/аргумента (например, вместо тонн передаются килограммы).

3. Не изменяет ли метод аргументы, являющиеся только входными?
4. Согласуются ли определения глобальных переменных во всех использующих их методах?

7. Ошибки ввода-вывода

Сводный список вопросов таков:

1. Правильны ли атрибуты файлов? Не происходит ли запись в файлы read-only?
2. Соответствует ли формат спецификации операторам ввода-вывода?

Не читаются ли строки вместо байт?

3. Соответствует ли размер буфера размеру записи?
4. Открыты ли файлы перед их использованием?
5. Обнаруживаются ли признаки конца файла?
6. Обнаруживаются ли ошибки ввода-вывода? Правильно ли трактуются ошибочные состояния ввода-вывода?
7. Существуют ли какие-нибудь текстовые ошибки в выходной информации?

Существуют ли смысловые или грамматические ошибки в тексте, выводимом программой на печать или экран дисплея? Все сообщения программы должны быть тщательно проверены.

2.5. Контрольные вопросы и задания

1. Дайте определение термина «ошибка».
2. Приведите классификацию ошибок по времени их появления.
3. Приведите классификацию ошибок по степени нарушения логики.
4. Какие ошибки (в разных классификациях) бывают в программах на языке C++ и когда они появляются?
5. Какие языки обнаруживают ошибки структурного набора?
6. Определите вид ошибки: if((x>3) && (x<2)) …
7. Какие типовые ошибки встречаются в программах?
8. В чем заключается сущность инспекции?

9. Какие этапы включает метод сквозного просмотра программы?
10. Приведите пример ошибки обращения к данным.
11. Приведите пример ошибки описания данных.
12. Приведите пример ошибки интерфейса.
13. Приведите пример ошибки передачи управления.
14. Приведите пример ошибки при сравнениях.
15. Приведите пример ошибки вычисления.
16. Приведите пример ошибки ввода-вывода.