



***Презентация по 4 и 5  
главам книги Канера  
«Тестирование  
программного  
обеспечения»***

Сидорина Анастасия, 332 группа математико-механического факультета СПбГУ



# Глава 4

Программные ошибки.

# Назначение главы

- Главной задачей тестировщика является поиск и документирование ошибок.

- Найденные ошибки исправляются; решаются проблемы, описанные тестировщиком, т.е. улучшается качество программного продукта.

- В этой главе мы определим понятия качества и программных ошибок, рассмотрим, какие бывают программные ошибки и как они классифицируются.

# ***Качество***

● Если клиент предоставляет подробную спецификацию с описанием своих требований, то **качество** будет означать **точное соответствие спецификации** клиента.

● Однако чаще всего критерием качества служит то, насколько пользователи **удовлетворены программным продуктом** и **сопутствующими услугами компании**.

# Качество(продолжение)

- Еще одной составляющей качества является **надежность программного продукта**. Она тем выше, чем реже в нем происходят сбои. Также качество нельзя назвать высоким, если программа не выполняет то, что пользователь считает важным.
- **Итак, качество программы определяется:**
  1. возможностями, благодаря которым она понравится пользователю;
  2. недостатками, которые вынуждают пользователя приобрести другую программу.
- Главное, что тестировщик может сделать для улучшения качества программы, — это выявить ее **недостатки**, **сбои** в её работе и **явные ошибки**.

# Программные ошибки

## Определения:

- Расхождение между программой и ее спецификацией является ошибкой тогда, и только тогда, когда спецификация существует и она правильна.
- Если программа не делает того, чего пользователь от нее вполне обоснованно ожидает, значит, налицо программная ошибка (Майерс).
- Не существует ни абсолютного определения ошибок, ни точного критерия наличия их в программе. Можно лишь сказать, насколько программа не справляется со своей задачей, — это исключительно субъективная характеристика (Бейзер).

# **Категории программных**

## **ошибок**

**Ошибки пользовательского интерфейса:**

- **Функциональность.**

Функциональные недостатки имеют место, если программа не делает того, что должна, выполняет одну из своих функций плохо или не полностью.

- **Взаимодействие программы с пользователем.**

Насколько сложно пользователю разобраться в работе с программой? Как обстоит дело с экранными инструкциями и подсказками?

- **Организация программы.**

Нет ли в программе непонятных команд? Какие ошибки чаще всего делает пользователь, на что он тратит больше всего времени и почему?

# Категории программных ошибок

## (продолжение)

- **Пропущенные команды.**

Не заставляет ли программа выполнять некоторые действия странным или неестественным способом? Допускает ли она хотя бы некоторую степень настройки?

- **Производительность.**

В интерактивном программном обеспечении очень важна скорость. Плохо, если у пользователя создается впечатление, что программа работает медленно.

- **Выходные данные.**

Получаете ли вы то, что хотите? Правильно ли формируются отчеты, наглядны ли диаграммы и достаточно ли отчетливо они выглядят на бумаге? Сохраняются ли данные в формате, доступном и для других аналогичных программ?

# **Категории программных ошибок**

## **(продолжение)**

### **Обработка ошибок.**

В процедуре обработки ошибок тоже часто встречаются ошибки. Кроме того, правильно определив ошибку, программа не всегда выдает о ней достаточно информативное сообщение.

### **Ошибки, связанные с обработкой граничных условий.**

Любой аспект работы программы, к которому применимы понятия  $>$  или  $<$ , раньше или позже, первый или последний, обязательно должен быть проверен на границах диапазона.

### **Ошибки вычислений.**

Одними из самых распространенных среди математических ошибок являются ошибки округления. Также к этой категории относятся и ошибки, вызванные неправильным выбором алгоритма.

# Категории программных ошибок (продолжение)

## ● Начальное и последующие состояния.

Бывает, что сбой программы происходит только однажды — при самом первом обращении к этой функции.

Насколько корректно поведет себя программа если пользователь вернется к исходным данным и изменит их?

## ● Ошибки управления потоком.

Если по логике программы вслед за первым действием должно быть выполнено второе, а она выполняет третье, значит, в *управлении потоком* допущена ошибка.

## ● Ошибки передачи или интерпретации данных.

Некоторые данные могут передаваться между модулями много раз, и на каком-то этапе они могут быть разрушены.

Изменения, внесенные в одной из частей программы, могут потеряться или достичь не всех частей системы, где они важны.

# **Категории программных ошибок**

## **(продолжение)**

### **Ситуация гонок.**

1. Предположим, в системе ожидаются два события, А и Б. Первым может произойти любое из них.

2. Если первым произойдет событие А, выполнение программы продолжится, а если первым наступит событие Б, то в работе программы произойдет сбой.

3. Программист полагал, что первым всегда должно быть событие А, и не ожидал, что Б может выиграть гонки.

Такие ошибки наиболее типичны для систем, где параллельно выполняются взаимодействующие процессы и потоки, а также для многопользовательских систем реального времени.

# Категории программных ошибок

(продолжение)

## Перегрузки.

1. Сбои могут происходить из-за интенсивной и длительной эксплуатации, нехватки памяти или отсутствия других необходимых ресурсов.

2. Соответствуют ли реальные возможности и требования программы к ресурсам ее спецификации и как программа себя поведет при перегрузках?

## Аппаратное обеспечение.

1. Программы могут посылать устройствам неверные данные, игнорировать их сообщения об ошибках, пытаться использовать устройства, которые заняты или вообще отсутствуют.

2. Даже если нужное устройство просто сломано, программа должна понять это, а не сбойть при попытке к нему обратиться.

# **Категории программных ошибок**

## **(продолжение)**

- **Контроль версий.**

Версии всех составляющих проекта обязательно должны централизованно контролироваться.

Необходима правильность появляющихся на экране сообщений об авторских правах, названии и номере версии программного продукта.

- **Документация.**

Это часть программного продукта, и если она плохо написана, пользователь может подумать, что и сама программа не намного лучше.

- **Ошибки тестирования.**

Иногда ошибки тестировщика отражают проблемы пользовательского интерфейса: если программа заставляет пользователя делать ошибки, значит, с ней что-то не так.



# **Глава 5**

## **Документирование и анализ ошибок.**

# ***Назначение этой главы***

- Вероятность исправления ошибки зависит от того, как ее задокументировать.
- Эта глава учит составлять отчеты, позволяющие наиболее эффективно взаимодействовать с руководителем проекта и программистом.
- Документированию ошибок необходимо уделять серьезное внимание: отчет должен составляться быстро, но при этом его содержание должно способствовать решению проблемы.

***Целью составления отчета об ошибке является ее исправление.***

Если вы хотите, чтобы найденная ошибка была быстро и надежно исправлена:

- **Объясните, как воспроизвести ошибку или проблемную ситуацию.**

Программисты игнорируют отчеты об ошибках, которых не могут увидеть своими глазами.

- **Тщательно проанализируйте ошибку, чтобы описать ее предельно кратко.**

Слишком пространное описание проблемы затрудняет понимание ее сути.

- **Составьте полный, понятный и непротиворечивый отчет.**

Если отчет путает программиста или недостаточно отчетливо составлен, он не послужит хорошей мотивацией для устранения ошибки.

НАЗВАНИЕ КОМПАНИИ \_\_\_\_\_

КОНФИДЕНЦИАЛЬНО

ОТЧЕТ О ПРОБЛЕМЕ № \_\_\_\_\_

ПРОГРАММА \_\_\_\_\_

ВЫПУСК \_\_\_\_\_

ВЕРСИЯ \_\_\_\_\_

ТИП ОТЧЕТА (1-6) \_\_\_\_\_

СТЕПЕНЬ ВАЖНОСТИ (1-3) \_\_\_\_\_

ПРИЛОЖЕНИЯ (Д/Н) \_\_\_\_\_

- 1 - Ошибка кодирования
- 2 - Ошибка проектирования
- 3 - Предложение
- 4 - Расхождение с документацией
- 5 - Взаимодействие с аппаратурой
- 6 - Вопрос

- 1 - Фатальная
- 2 - Серьезная
- 3 - Незначительная

Если да, какие:

### ПРОБЛЕМА

МОЖЕТЕ ЛИ ВЫ ВОСПРОИЗВЕСТИ ПРОБЛЕМНУЮ СИТУАЦИЮ? (Д/Н) \_\_\_\_\_

ПОДРОБНОЕ ОПИСАНИЕ ПРОБЛЕМЫ И КАК ЕЕ ВОСПРОИЗВЕСТИ

ПРЕДЛАГАЕМОЕ ИСПРАВЛЕНИЕ (НЕОБЯЗАТЕЛЬНО)

ОТЧЕТ ПРЕДСТАВЛЕН СОТРУДНИКОМ \_\_\_\_\_

ДАТА \_\_/\_\_/\_\_

СЛЕДУЮЩИЕ ГРАФЫ ПРЕДНАЗНАЧЕНЫ ТОЛЬКО ДЛЯ РАЗРАБОТЧИКОВ

ФУНКЦИОНАЛЬНАЯ ОБЛАСТЬ \_\_\_\_\_

ОТВЕТСТВЕННЫЙ \_\_\_\_\_

### КОММЕНТАРИИ

СОСТОЯНИЕ (1-2) \_\_\_\_\_

ПРИОРИТЕТ (1-5) \_\_\_\_\_

- 1 - Открыто
- 2 - Закрыто

РЕЗОЛЮЦИЯ (1-9) \_\_\_\_\_

ИСПРАВЛЕННАЯ ВЕРСИЯ \_\_\_\_\_

- 1 - Рассматривается
- 2 - Исправлено
- 3 - Не воспроизводится
- 4 - Отложено
- 5 - Соответствует проекту
- 6 - Не может быть исправлено

- 7 - Отозвано составителем
- 8 - Нужна дополнительная информация
- 9 - Не согласен с предложением

РАССМОТРЕНО \_\_\_\_\_

ДАТА \_\_/\_\_/\_\_

ПРОКОНТРОЛИРОВАНО \_\_\_\_\_

ДАТА \_\_/\_\_/\_\_

СЧИТАТЬ ОТЛОЖЕННЫМ (Д/Н)

# *Структура отчета о проблеме*

- **Программа.**

Если несколько программ, то следует указывать, в какой из них обнаружена ошибка.

- **Выпуск и версия.**

Наличие в отчетах номеров версий тестируемых программ позволяет избежать путаницы с уже исправленными и повторно выявленными ошибками.

# Структура отчета о проблеме (продолжение)

## Тип отчета.

- Ошибка кодирования.

Программа ведет себя не так, как должна по мнению тестировщика. Пример:  $2 + 2 = 3$ .

- Ошибка проектирования.

Программа соответствует проектной документации, но в определенном вопросе тестировщик с ней не согласен.

- Предложение.

Описывается идея, реализация которой, по мнению тестировщика, может улучшить программу.

- Расхождение с документацией.

Программа ведет себя не так, как описано в руководстве или интерактивной справке.

- Взаимодействие с аппаратурой.

Эта проблема связана с неудачным взаимодействием программы и аппаратного обеспечения.

- Вопрос.

Программа делает что-то, чего тестировщик не ожидает или не понимает.

# **Структура отчета о проблеме (продолжение)**

## **Степень важности.**

Тестировщик указывает насколько серьезна выявленная проблема. Если в программе много мелких ошибок, необходимо составить о них единый отчет.

## **Приложения.**

К отчету о найденной ошибке можно приложить все, что поможет программисту разобраться в ситуации.

# Структура отчета о проблеме (продолжение)

- **Проблема.**

В этой графе суть проблемы формулируется очень коротко — в одной-двух строчках.

В графе **Проблема** следует описать возникающую проблему, а не рассказывать, как воспроизвести ошибку.

- **Можете ли вы воспроизвести проблемную ситуацию?**

Ответом может быть **Да**, **Нет** или **Не всегда**. Если с повторением ситуации возникли сложности, лучше отложить составление отчета до тех пор, пока дело не прояснится.

## ***Структура отчета о проблеме (продолжение)***

- **Подробное описание проблемы и как её воспроизвести.**

Нужно написать, в чем состоит проблема, и почему вы считаете, что что-то не в порядке. Нужно описать все шаги, включая сообщения об ошибке.

**Никогда не игнорируйте проблему только потому, что она не воспроизводится!**

- **Предлагаемое исправление.**

Эта графа отчета не является обязательной. Если решение проблемы очевидно или у вас нет конкретного предложения, оставьте ее пустой.

- **Отчет представлен сотрудником.**

Обязательно укажите здесь свою фамилию. Если у программиста возникнут вопросы, он должен знать, к кому обратиться.

- **Дата.**

В этой графе следует указать дату обнаружения проблемы

# Структура отчета о проблеме (продолжение)

Следующие графы отчета предназначены только для команды разработчиков.

- **Функциональная область.**

В этой графе указывается, к какой категории относится выявленная проблема.

- **Поручено.**

Кто из сотрудников отвечает за решение описанной проблемы.

- **Комментарии.**

1. Если отслеживание ошибок и их исправления ведется на бумаге, эта графа используется программистом. Он может *коротко* записать, почему отчет отложен или как решена проблема.

2. В многопользовательских системах каждый, кто имеет доступ к отчету, может внести собственный комментарий.

# Структура отчета о проблеме (продолжение)

## ● Состояние.

В поле **Состояние** только что написанного отчета записывается **Открыто**. После исправления ошибки, значение этого поля изменяется на **Закрыто**.

## ● Приоритет.

Ошибки исправляются в порядке их приоритета.

- (1) Исправить немедленно — ошибка задерживает работу других сотрудников.
- (2) Исправить как можно быстрее.
- (3) Исправить в текущей версии (альфа, бета и т.д.).
- (4) Исправить до выхода окончательной версии.
- (5) Исправить, если возможно.
- (6) Не обязательно — сделайте, как посчитаете нужным.

Графу **Приоритет** имеет право заполнять *только* руководитель проекта, а графу **Степень важности** — только составитель отчета или руководитель группы тестирования.

# Структура отчета о проблеме

## (продолжение)

### ● Резолюция и Исправленная версия.

1. В графе **Резолюция** определяется текущее состояние вопроса или принятое по нему решение.

2. Если в ответ на данный отчет в программу были внесены изменения, в графе **Исправленная версия** программист указывает номер исправленной версии программы.

### Варианты резолюции:

(1) *рассматривается.*

(2) *исправлено.*

(3) *не воспроизводится.*

(4) *отложено.*

(5) *соответствует **проекту.***

(6) *отозвано составителем.*

(7) *нужна дополнительная информация.*

(8) *не согласен с предложением.*

(9) *дубликат.*

# Структура отчета о проблеме

(продолжение)

- **Подписи.**

В графе **Рассмотрено** должна стоять подпись сотрудника, решившего проблему или подпись его руководителя. В графе **Проконтролировано** ставит свою подпись тестировщик, подтверждающий, что отчет можно закрыть.

- **Считать отложенным.**

1. Исправление ошибки или решение проблемы может быть по решению руководителя отложено до следующего выпуска программы. Так можно поступить с любой ошибкой, которую по определенным причинам уже нет времени или возможности исправить.

***2. Некоторые программисты намеренно прячут под спорными или неверными резолюциями легко воспроизводимые и исправимые ошибки, чтобы скрыть от руководства халтурную работу или то, что они не укладываются в сроки.***

# Каким должен быть отчет о

## проблеме

- **Реальный документ.**

1. Если программист не исправит ошибку сразу же, как только вы о ней расскажете, она должна быть описана. Но даже если исправит сразу, исправления нужно будет протестировать(отчет).

2. Описания ошибок можно не записывать: работа программистов задолго до начала официального тестирования. Большинство обнаруженных на этом этапе проблем к началу формального тестирования уже давно не будет.

- **Нумерация.**

Отчет о проблеме должен иметь уникальный номер. В базе данных этот номер должен быть *ключевым полем таблицы отчетов*, т.е. однозначным идентификатором отчета в системе.

# Каким должен быть отчет о проблеме (продолжение)

- **Простота.**

В каждом отчете должна быть описана только одна проблема. Даже если пять проблем кажутся тесно связанными, все равно следует составить о НИХ пять разных отчетов.

- **Понятность.**

Чем понятнее отчет, тем больше вероятность, что описанная в нем ошибка будет исправлена.

- **Воспроизводимость.**

Если вы знаете, как воспроизвести ситуацию, опишите этот процесс шаг за шагом, чтобы программист смог сразу его повторить. А если нет, прямо напишите об этом в отчете.

# *Каким должен быть отчет о проблеме (продолжение)*

- **Разборчивость.**

Рукописный отчет должен быть разборчивым. Если отчет вводится в компьютер, это все равно нужно сделать аккуратно.

- **Беспристрастность.**

Ни в коем случае нельзя допускать оценок работы программиста.

Для решения вопроса начальство может установить цензуру.

# Анализ воспроизводимой

## ошибки

*Если проблему можно воспроизвести, то:*

- Тестировщик может описать, как перевести программу в определенное состояние. По его описанию это сделать может любой знакомый с ней человек.
- Тестировщик может описать конкретные действия, которые в указанном состоянии программы приводят к проявлению проблемы.

### **Главные цели анализа таковы:**

- Выявить все наиболее серьезные последствия проблемы.
- Найти простейший и кратчайший путь ее воспроизведения.
- Найти альтернативные действия, приводящие к такому же результату.
- Выявить связанные проблемы.

# *Анализ воспроизводимой ошибки (продолжение)*

## **Наиболее серьезные последствия проблемы.**

Чтобы привлечь к проблеме внимание, нужно представить ее более серьезной.

Сбой программы — это:

- ее переход в непредусмотренное программистом состояние;
- передача управления блоку обработки ошибки.

# **Анализ воспроизводимой ошибки**

**(продолжение)**

## **Простейший и кратчайший путь воспроизведения ситуации.**

Бывает, что обнаруженная ошибка возникает в ответ на сложную последовательность нестандартных или редко выполняемых действий пользователя.

- Если ошибка понятна и ее легко исправить, это будет сделано.
- Если для исправление ошибки требуется много времени и усилий или программисту кажется, что это так, он возьмется за работу с большой неохотой.
- Если проблема проявляется при самых типичных действиях пользователя программы, руководство будет заинтересовано в ее исправлении.
- Если кажется, что недостатка практически никто не заметит, в очереди на исправление он будет последним.

# Анализ воспроизводимой ошибки (продолжение)

- **Альтернативный способ демонстрации ошибки.**

Бывает, что найти более простой способ воспроизведения проблемной ситуации так и не удастся. Можно поискать другие действия, приводящие к проявлению этой же проблемы.

Два разных способа вызвать ошибку —уже более серьезный сигнал тревоги.

- **Связанные проблемы.**

Проверим, нет ли в программе фрагмента, в котором может проявиться такая же проблема. Возможно, пользователь в нескольких ее режимах выполняет сходные действия.

# Методика анализа воспроизводимой ошибки.

- **Выделение критического момента.**

Обнаружив ошибку, тестировщик видит только симптом, но не знает причины.

**Симптомы ошибок:**

- *Сообщения об ошибках.*

Выясните, в какой момент появляется сообщение об ошибке и почему, сверьте его с перечнем сообщений об ошибках, перечисленных в документации к программе.

- *Задержки в обработке данных.*

Всегда обращайтесь внимание на подозрительно долгие задержки и тщательно проверяйте результирующие данные.

- *Сдвинутый текст.*

# Методика анализа воспроизводимой ошибки

## (продолжение)

- *Мигание и обновление экрана.*

Если экран неожиданно мигнул, то возможно он был обновлен подпрограммой обработки ошибок. Это может быть первое, что она сделает, а остальные ее действия могут проявиться позднее или не проявиться вовсе.

- *Перемещение курсора.*

Это может быть сделано процедурой обработки ошибок, а может быть и результатом сбоя в работе самой программы.

- *Несколько курсоров.*

- *Повторяющиеся или пропущенные символы.*

- *Горящий индикатор активности устройства, которое не должно участвовать в работе.*

Программист записал данные не по тому адресу, и вместо определенного места памяти они попали на диск.

# Методика анализа воспроизводимой ошибки (продолжение)

## Отслеживание действий программы.

1. Можно воспользоваться отладчиком исходного кода: какой процесс в данный момент активен, какой объем памяти компьютер использует, насколько заполнен стек и т.д.

2. Распечатка экранов программы и изменений в файлах данных.

3. Если содержимое экрана очень быстро меняется, можно попробовать поработать на менее скоростном компьютере или эксплуатировать программу в многопользовательской среде с повышенной нагрузкой.

# Методика анализа воспроизводимой ошибки

**(продолжение)**

- **Выявив критический шаг, тщательно протестируйте его последствия.**

1. Если пользователь выполняет шаги А, Б, В, а на шаге В что-то происходит не так, то ошибка на шаге Б.

2. Поменяйте последовательность действий, посмотрите, как поведет себя программа, если после шага Б выполнить не В, а Г. Возможно, обнаружится более серьезная проблема, чем та, которую вы встретили вначале.

- **Поищите дальнейшие ошибки.**

За найденной ошибкой возможно последуют другие, и информация об этих последствиях первой ошибки может помочь программисту в ее **поиске** и **исправлении**.

# Методика анализа воспроизводимой ошибки

**(продолжение)**

- **Варьируйте последовательность действий.**

Оставьте в отчете только те действия, которые действительно необходимы для воспроизведения ситуации.

- **Проверьте, имелась ли такая же ошибка в предыдущих версиях программы.**

Если найденная вами ошибка появилась только в определенной версии программы, значит ошибка появилась в результате конкретных изменений.

- **Проверьте, не зависит ли поведение программы от конфигурации системы.**

Попробуйте добавить память или уменьшить ее объем. Проверьте, как программа работает в сети. Попробуйте выгрузить все лишние программы.

# Поиск способа воспроизведения ошибки

- Ошибка воспроизводима, если ее можно увидеть, выполнив описанные в отчете действия. Задача составителя — объяснить, какие действия нужно выполнять для воспроизведения ошибки и рассказать, в чем она состоит.
- Если тестировщик не знает, как получил ошибку, то следует записать все, что вы помните: в чем состояла ошибка и что вы делали, перед тем как ее увидели.
- Многие тестировщики делают видеозаписи своей работы; пользуются программами перехвата клавиатурного и иного ввода, с помощью которых впоследствии можно воспроизвести свои действия.

# **Направления поиска источника ошибки.**

- **Повышенная нагрузка.**

Обычно тестировщик выполняет привычный тест очень быстро. Столкнувшись с ошибкой, он повторяет выявивший ее тест медленнее и внимательнее.

Нужно выполнить тест с обычной скоростью несколько раз; увеличить нагрузку на компьютер или перейти на менее скоростную машину.

- **Пропущенные детали.**

Тестировщик может забыть, что именно он делал, или упустить какую-нибудь мелкую деталь, являющуюся ключевой.

# **Направления поиска источника ошибки (продолжение)**

- **Ошибка пользователя: вы сделали вовсе не то, что думаете.**

Тестировщик может ошибиться; например, если исчезли нужные данные, это может означать, что вы их случайно удалили. Но предполагать собственную ошибку следует **в последнюю очередь**.

- **Ошибка с разрушительными последствиями.**

Бывает, что последствия ошибки настолько разрушительны, что ее невозможно сразу повторить, т.е. чтобы приступить к воспроизведению ошибки, придется сначала восстановить работоспособность системы.

**Никогда не работайте с исходными данными — только с копиями!**

# Направления поиска источника ошибки (продолжение)

- **Ошибка, зависящая от объема памяти.**

Сбой программы может происходить при условиях, связанных с типом, объемом и структурой доступной памяти. В этом случае полезно вставить в программу отладочные сообщения об объеме доступной памяти при ее загрузке и выполнении.

- **Ошибка, проявляющаяся только при первом запуске.**

1. Если при первом запуске программы инициализационные файлы не содержат нужной информации, программа может вести себя непредсказуемо. При выходе программа сохранит в этих файлах корректные данные, и дальше все пойдет хорошо.

2. Еще одной сходной ошибкой может быть неправильная интерпретация программой собственной неинициализированной памяти.

# Направления поиска источника ошибки (продолжение)

- **Ошибка, связанная с разрушенными данными.**  
Программа может обнаружить разрушение данных и выдать вполне корректное сообщение об ошибке, а может повести себя совершенно непредсказуемо.
- **Ошибка, являющаяся побочным эффектом другой проблемы.**  
Если в процедуре обработки ошибок имеется ошибка, восстановление после очередного сбоя может пройти неудачно.
- **Нерегулярные аппаратные сбои.**  
Бывает, что из-за плохого контакта или случайного колебания напряжения происходит единовременный сбой, который больше не повторяется.  
На аппаратный сбой ошибку следует списывать **в самую последнюю очередь.**

# Направления поиска источника ошибки (продолжение)

- **Дата и время.**

В любой программе, работающей с датами и временем, следует внимательно проверить значения, попадающие на границу суток, недели, месяца, года, високосного года и столетия.

- **Зависимость от ресурсов.**

1. В программе должны быть предусмотрены действия, выполняемые в случае отсутствия необходимых ресурсов.

2. Для воспроизведения ошибки потребуется восстановить состояние среды выполнения — снова запустить программы, занимающие память, принтер, коммуникационный порт и т.п.

# **Направления поиска источника ошибки (продолжение)**

- **Пауза между ошибкой и ее проявлением.**

Возможно, ошибка должна будет повториться много раз, и видимый результат может проявиться при выполнении много раз проверенной подпрограммы. Типичным примером подобной ситуации является переполнение стека.

- **Особые фрагменты кода.**

Тестируя программу извне, вы не знаете, какие критические точки и граничные условия присутствуют в ее коде.

- **Кто-то поэкспериментировал с вашим компьютером.**

Оставляя компьютер включенным, вы всегда рискуете, вернувшись, найти его в другом состоянии.