Верификация программного обеспечения

Методы тестирования

Родионова Алиса Витальевна

Testing Methods

- Black Box
  - Equivalence Class Testing
  - Boundary Value Testing
  - State Transition Testing
- White Box
  - Code Coverage

Equivalence classes

- Если от выполнения двух тестов ожидается один и тот же результат, они могут считаться *эквивалентными*.

- Группа тестов представляет собой *класс эквивалентности*, если выполняются следующие условия:

  - Все тесты предназначены для выявления одной и той же ошибки

  - Если один тест выявит ошибку, все остальные, скорее всего, тоже это сделают

  - Если один из тестов не выявит ошибку, то и все остальные, скорее всего, этого не сделают

Equivalence classes Example

- Программа должна принимать числа от 1 до 99
  - Числа от 1 до 99
  - Числа меньше 1
  - Числа больше 99

Equivalence classes

- Тесты включают значения одних и тех же входных данных
- Для их проведения выполняются одни и те же операции программы
- В результате всех тестов формируются значения одних и тех же выходных данных
- Либо ни один из тестов не вызывает выполнение блока обработки ошибок программы, либо выполнение этого блока вызывается всеми тестами группы

Equivalence classes search

- ✔ Не забывайте о классах, охватывающих заведомо неверные или недопустимые входные данные
- ✔ Организуйте формируемый перечень классов в виде таблицы или плана
- ✔ Определите диапазоны числовых значений
- ✔ Для полей или параметров, принимающих фиксированные перечни значений, выясните, какие из значений входят в перечень
- ✔ Проанализируйте возможные результаты выбора из списков и меню

Equivalence classes search

- ✔ Поищите переменные, значения которых должны быть равными
- ✔ Поищите классы значений, зависящих от времени
- ✔ Выявите группы переменных, совместно участвующих в определенных вычислениях, результат которых ограничивается конкретным набором или диапазоном значений
- ✔ Посмотрите, на какие действия программа отвечает эквивалентными событиями
- ✔ Продумайте варианты операционного окружения

Equivalence classes Example

- Программа должна принимать числа от 1 до 99
  - Числа от 1 до 99
  - Числа меньше 1
  - Числа больше 99
  - Нечисловая информация

Equivalence classes Table

Входное/ выходное событие	Допустимые классы эквивалентности	Недопустимые классы эквивалентности
Ввод числа	•Числа от 1 до 99	•0 •>99 •Выражение, результат которого – недопустимое число (5-5=0) •Отрицательные числа •Буквы и другие нечисловые символы
Ввод первой буквы имени	•Первый символ является заглавной буквой •Первый символ является прописной буквой	•Первый символ не является буквой
Рисовани	•От одной точки до	•Отсутствие рисунка

Equivalence classes Plan

1. Ввод числа

1.1 Допустимые варианты

1.1.1 Числа от 1 до 99

1.2 Недопустимые варианты

1.2.1 0

1.2.2 > 99

1.2.3 Выражение, результатом которого является недопустимое число, например, **5 - 5**, результат которого равен **0**.

1.2.4 Отрицательные числа

1.2.5 Буквы и другие нечисловые символы

1.2.5.1 Буквы

1.2.5.2 Арифметические операции, такие как +, * , -

1.2.5.3 Остальные нецифровые символы

1.2.5.3.1 Символы с ASCII-кодами, меньшими кода нуля

1.2.5.3.2 Символы с ASCII-кодами, большими кода девятки

Equivalence classes Plan

2. Ввод первой буквы имени

2.1 Допустимые варианты

2.1.1 Первый символ является заглавной буквой

2.1.2 Первый символ является прописной буквой

2.2 Недопустимые варианты

2.2.1 Первый символ не является буквой

2.2.1.1 Первый символ имеет ASCII-код, меньший кода буквы "A"

2.2.1.2 Первый символ имеет ASCII-код, лежащий между кодами букв "Я" и "а"

2.2.1.3 Первый символ имеет ASCII-код, больший кода буквы "я"

Equivalence classes Plan

3. Рисование прямой

3.1 Допустимые варианты

3.1.1 От одной точки до четырех сантиметров длиной

3.2 Недопустимые варианты

3.2.1 Отсутствие рисунка

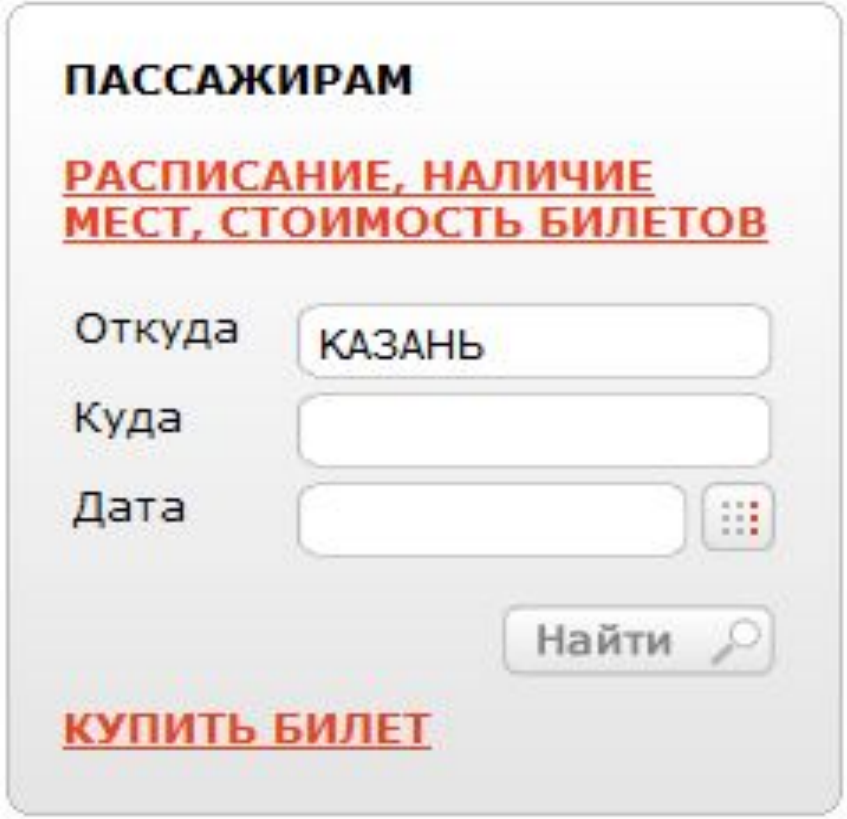
3.2.2 Длиннее четырех сантиметров

3.2.3 Линия не является прямой

3.2.3.1 Что это может быть??? Кривая? Окружность?

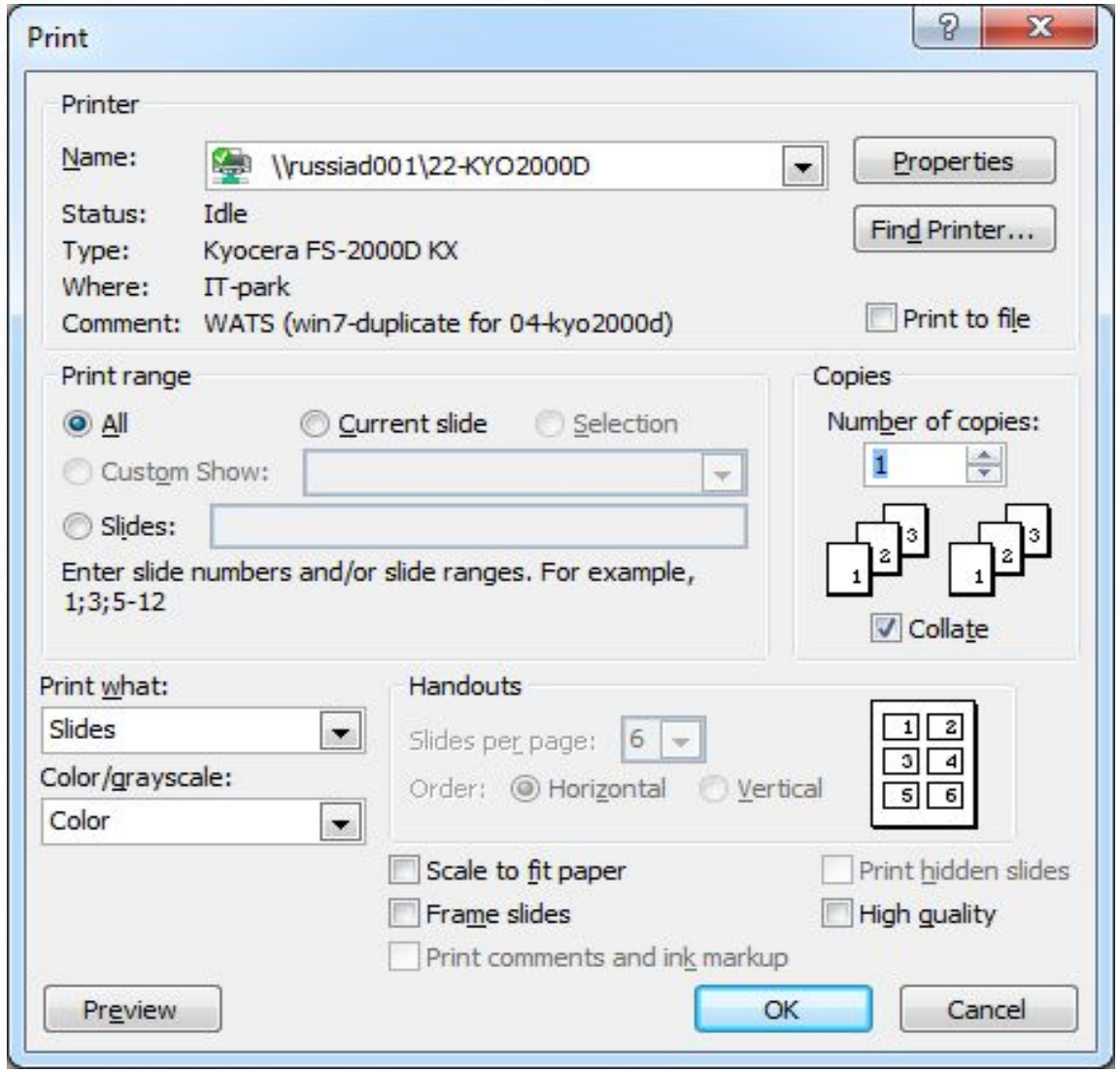
Equivalence classes

- Для полей или параметров, принимающих фиксированные перечни значений, выясните, какие из значений в них входят



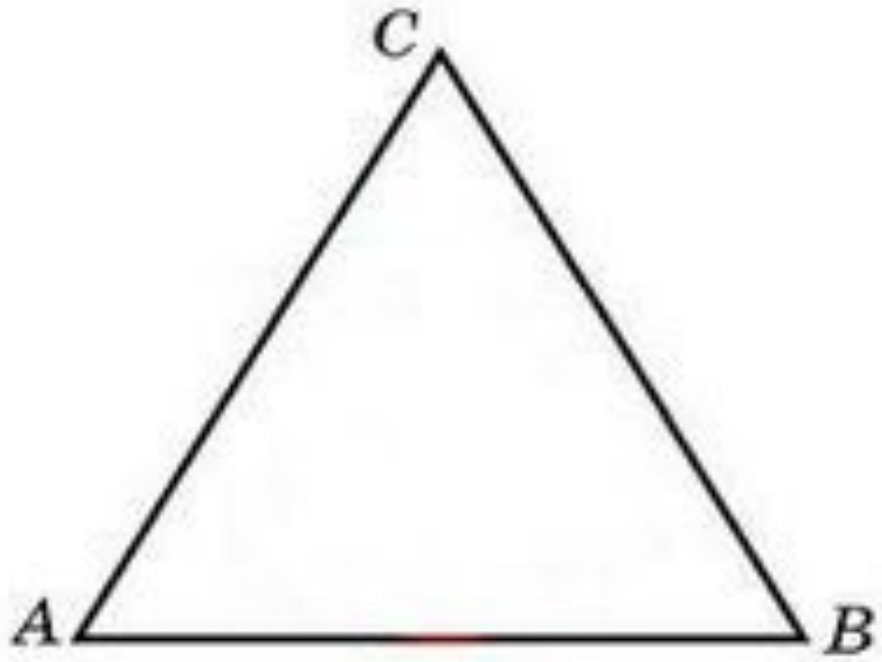
Equivalence classes

- Поищите классы значений, зависящих от времени



Equivalence classes

- Выявите группы переменных, совместно участвующих в определенных вычислениях, результат которых ограничивается конкретным набором или диапазоном значений



Boundary values

- Необходимо протестировать каждую границу класса эквивалентности, причем с обеих сторон
- Программа, которая пройдет эти тесты, скорее всего, пройдет и все остальные, относящиеся к данному классу.

Boundary values Examples

Входные параметры	Граничные значения
Допустимы значения от 1 до 99	•Допустимые данные: 1 и 99 •Недопустимые данные: 0 и 100
Программа выписывает чеки на суммы от 1$ до 99$	•Чек на отрицательную сумму •0$ •100$
Программа ожидает заглавную букву	•A и Z •@ (код перед кодом символа A) •] (код после кода символа Z) •a и z
Программа рисует линии длиной от одной точки до 4 сантиметров	•Точка и линия длиной 4 см •Линия нулевой длины
Сумма входных значений должна равняться 180	•Сумма 179 •Сумма 180 •Сумма 181

Boundary values Techniques

Шаги для тестирования граничных значений

- Определите классы эквивалентности
- Определите границы каждого класса эквивалентности
- Протестируйте следующие значения:
  - ✔ На границе
  - ✔ На одно значение меньше границы
  - ✔ На одно значение больше границы

State Transition Testing

- ✔ Протестируйте все наиболее вероятные последовательности действий пользователей
- ✔ Если возможно предположить, что действия пользователя в одном режиме могут воздействовать на представление данных или набор предоставляемых программой возможностей в другом режиме, протестируйте эту зависимость
- ✔ Кроме проведения самых необходимых тестов, стоит поработать с программой в произвольном режиме, случайным образом выбирая путь ее выполнения

Practice

Определите классы эквивалентности и граничные условия.

- Возраст – не менее 18

1) Возраст <18 – недопустимый класс
2) Возраст >=18 – допустимый класс
3) Некорректные символы – недопустимый класс

Протестируем:

1) 17, 18, 19
2) 1000, abc, $, 0, -1, @, 1$, …

Tasks

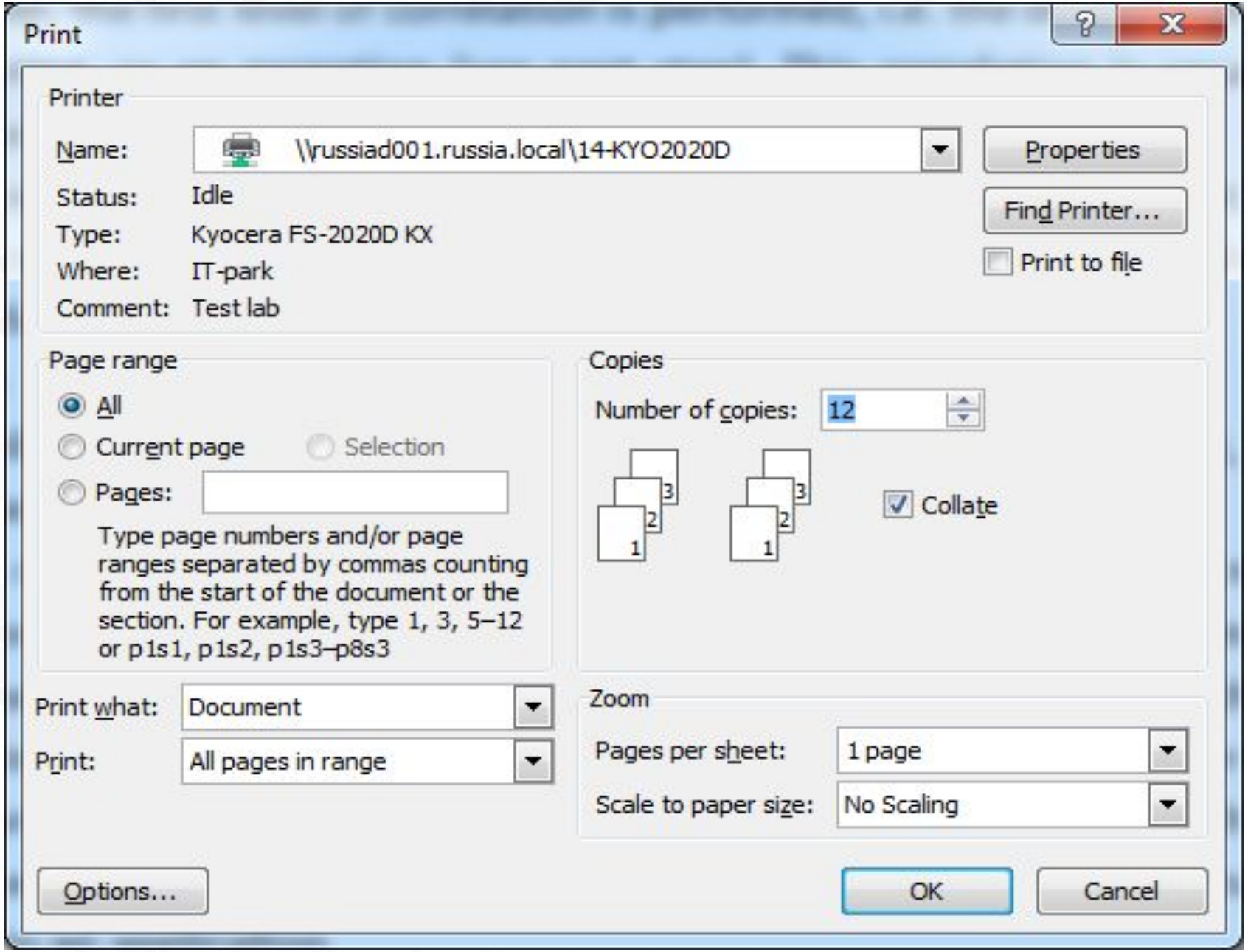
Определите классы эквивалентности и граничные условия.

1) Для различных видов данных:

a) Индекс – 6 цифр

b) Фамилия – 15 символов (буквы или пробелы, дефис)

2) Для всех полей окна MS Office Word File -> Print


Print
Printer
Name:
\\russiad001.russia.local\14-KYO2020D
Properties
Status:
Idle
Find Printer...
Type:
Kyocera FS-2020D KX
Where:
IT-park
Print to file
Comment:
Test lab
Page range
All
Current page
Selection
Pages:
Type page numbers and/or page ranges separated by commas counting from the start of the document or the section. For example, type 1, 3, 5–12 or p1s1, p1s2, p1s3–p8s3
Copies
Number of copies:
12
3
2
1
3
2
1
Collate
Print what:
Document
Print:
All pages in range
Zoom
Pages per sheet:
1 page
Scale to paper size:
No Scaling
Options...
OK
Cancel

State Transition Testing

Тестирование состояний переходов - тестирование методом черного ящика, в котором сценарии тестирования строятся на основе выполнения корректных и некорректных переходов состояний.

State Transition Testing

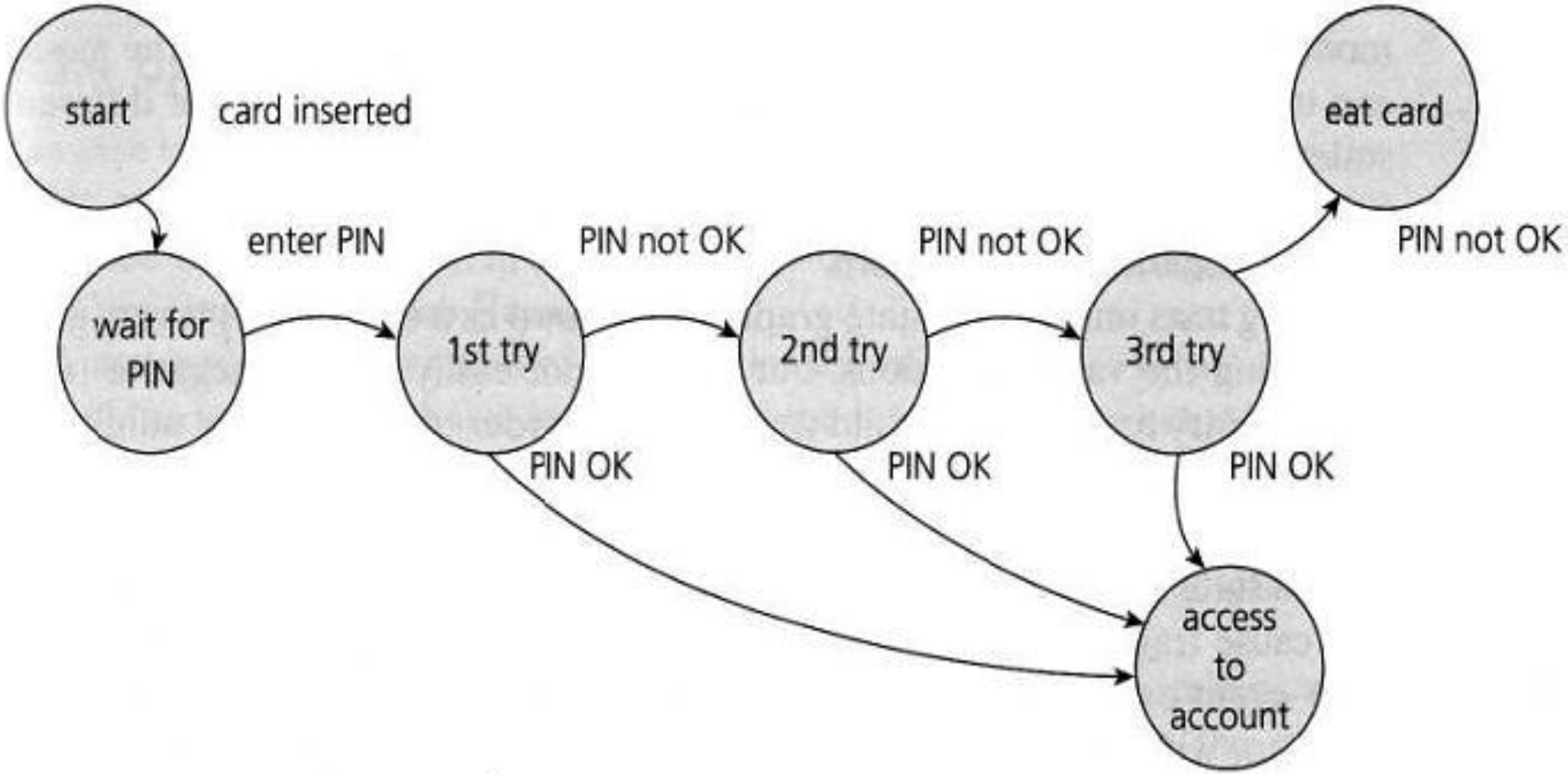


Диаграмма переходов состояний на примере банкомата

State Transition Testing

- ✔ Протестируйте все наиболее вероятные последовательности действий пользователей
- ✔ Если возможно предположить, что действия пользователя в одном режиме могут воздействовать на представление данных или набор предоставляемых программой возможностей в другом режиме, протестируйте эту зависимость
- ✔ Кроме проведения самых необходимых тестов, стоит поработать с программой в произвольном режиме, случайным образом выбирая путь ее выполнения

White box design techniques

Test coverage

- **Покрытие –** уровень, выражаемый в процентах, на который определенный элемент покрытия был проверен набором тестов.

- **Тестовое Покрытие** - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

- *Чем выше требуемый уровень тестового покрытия, тем больше тестов будет выбрано, для проверки тестируемых требований или исполняемого кода.*

Подходы к оценке и измерению тестового покрытия

- **Покрытие требований (Requirements Coverage)** – оценка покрытия тестами функциональных и нефункциональных требований к продукту путем построения матриц трассировки (traceability matrix).

- **Покрытие кода (Code Coverage)** – оценка покрытия исполняемого кода тестами путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

- **Тестовое покрытие на базе анализа потока управления** – оценка покрытия, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для

Различия и ограничения подходов

- **Различия**:
  Метод *покрытия требований* сосредоточен на проверке соответствия набора проводимых тестов требованиям к продукту.

  Анализ *покрытия кода* - на полноте проверки тестами, разработанной части продукта (исходного кода).

  Анализ *потока управления* - на прохождении путей в графе или модели выполнения тестируемых функций (Control Flow Graph).

- **Ограничения**:
  Метод оценки *покрытия кода* не выявит нереализованные требования, так как работает не с конечным продуктом, а с существующим исходным кодом.

  Метод *покрытия требований* может оставить непроверенными некоторые участки кода, потому что не учитывает конечную реализацию.

Requirements Coverage

Tcov = (Lcov/Ltotal) * 100%

Tcov - тестовое покрытие
Lcov - количество требований, проверяемых тест кейсами
Ltotal - общее количество требований

Code Coverage

Tcov = (Ltc/Lcode) * 100%

Tcov - тестовое покрытие
Ltc - кол-во строк кода, покрытых тестами
Lcode - общее кол-во строк кода.

Control Flow Testing

- **Тестирование потоков управления** (Control Flow Testing) - это одна из техник тестирования белого ящика, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Control Flow Testing

- Фундаментом для тестирования потоков управления является построение графов потоков управления (Control Flow Graph), основными блоками которых являются:

  - блок процесса - одна точка входа и одна точка выхода
  - точка альтернативы - одна точка входа, две и более точки выхода
  - точка соединения - две и более точек входа, одна точка выхода

Уровни тестового покрытия

Уровень	Название	Краткое описание
Уровень 0	--	“Тестируй все что протестируешь, пользователи протестируют остальное” На английском языке это звучит намного элегантнее: *“Test whatever you test, users will test the rest”*
Уровень 1	Покрытие операторов	Каждый оператор должен быть выполнен как минимум один раз.
Уровень 2	Покрытие альтернатив/ Покрытие ветвей	Каждый узел с ветвлением (альтернатива) выполнен как минимум один раз.
Уровень 3	Покрытие условий	Каждое условие, имеющее TRUE и FALSE на выходе, выполнено как минимум один раз.

Уровни тестового покрытия

Уровень	Название	Краткое описание
Уровень 4	Покрытие условий альтернатив	Тестовые случаи создаются для каждого условия и альтернативы
Уровень 5	Покрытие множественных условий	Достигается покрытие альтернатив, условий и условий альтернатив (Уровни 2, 3 и 4)
Уровень 6	«Покрытие бесконечного числа путей»	Если, в случае зацикливания, количество путей становится бесконечным, допускается существенное их сокращение, ограничивая количество циклов выполнения, для уменьшения числа тестовых случаев.
Уровень 7	Покрытие путей	Все пути должны быть проверены

White box design techniques

- Structure-based techniques serve two purposes: test coverage measurement and structural test case design.
- They are then used to design additional tests with the aim of increasing the test coverage.
- They can help ensure more breadth of testing, in the sense that test cases that achieve 100% coverage in any measure will be exercising all parts of the software from the point of view of the items being covered.

White box design techniques

Code Coverage

- ✔ Test coverage measures in some specific way the amount of testing performed by a set of tests.
- ✔ The basic coverage measure is

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

where the 'coverage item' is whatever we have been able to count and see whether a test has exercised or used this item.

White box design techniques

- There is danger in using a coverage measure. 100% coverage does *not mean* 100% tested!
- Coverage techniques measure only one dimension of a multi-dimensional concept.
- Two different test cases may achieve exactly the same coverage but the input data of one may find an error that the input data of the other doesn't.

White box design techniques

- ✔ Statement coverage
- ✔ Branch coverage
- ✔ Decision coverage

White box design techniques

Statement coverage

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

White box design techniques

- Studies and experience in the industry have indicated that what is considered reasonably through black-box testing may actually achieve only 60% to 75% statement coverage.

- Typical ad hoc testing is likely to be around 30% - this leaves 70% of the statements untested.

White box design techniques

```
READ A
READ B
IF A>B
  THEN C = 0
ENDIF
```

Сколько нужно провести тестов, чтобы получить 100% statement coverage?

1 тест. Например A=10, B=5

White box design techniques

Branch coverage

A decision is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL), or a CASE statement, where there are two or more possible exits or outcomes from the statement.

White box design techniques

Decision coverage

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\%$$

White box design techniques

- Functional testing may achieve only 40% to 60% branch coverage.
- Typical ad hoc testing may cover only 20% of the decisions, leaving 80% of the possible outcomes untested.

White box design techniques

- Decision coverage is stronger than statement coverage.
- 100% decision coverage always guarantees 100% statement coverage.

White box design techniques

```
READ A
READ B
C = A - 2 *B
IF C <0 THEN
   PRINT "C negative"
END IF
```

Сколько нужно провести тестов, чтобы получить 100% branch coverage?

2 теста. Например A=11, B=5 (C>=0)

A=10, B=15, (C<0)