Test Generation – Domain Partitioning

Andrea Polini

Fundamentals of Software Testing
MSc in Computer Science
University of Camerino

ToC

Software Requirements

Requirements Specification

- informal
- semi-formal
- formal

Depending on the degree of formality more or less automated strategies can be applied

The test selection problem

Challenge

Construct a test set $\mathscr{T} \subseteq \mathscr{D}$ that will reveal as many errors in *p* as possible (where $\mathscr{D}$ is the input domain and $\mathscr{T}$ is the set of tests) given the budget and temporal constraints imposed by the project

To give an idea...

Consider a procedure that has to manage data of an employee defined as follows:

- ID:`int` – three digit long from 001 to 999
- name:`string` – name is a 20 character long. Each characters belong to the set of 26 letters and space
- rate:`float` – rate varies from \$5 to \$10 per hour and in multiple of a quarter
- hoursWorked:`int` – hoursWorked varies from 0 to 60

Therefore:

$$999 \times 27^{20} \times 21 \times 61 \approx 5.42 \times 10^{34}$$

If time to execute a test is equal to 1*ns* total time to execute all test will be

The test selection problem

Challenge

Construct a test set $\mathscr{T} \subseteq \mathscr{D}$ that will reveal as many errors in *p* as possible (where $\mathscr{D}$ is the input domain and $\mathscr{T}$ is the set of tests) given the budget and temporal constraints imposed by the project

To give an idea...

Consider a procedure that has to manage data of an employee defined as follows:

- ID:`int` – three digit long from 001 to 999
- name:`string` – name is a 20 character long. Each characters belong to the set of 26 letters and space
- rate:`float` – rate varies from \$5 to \$10 per hour and in multiple of a quarter
- hoursWorked:`int` – hoursWorked varies from 0 to 60

Therefore:

$$999 \times 27^{20} \times 21 \times 61 \approx 5.42 \times 10^{34}$$

If time to execute a test is equal to 1*ns* total time to execute all test will be

Equivalence partitioning

How to . . .

using the equivalence partitioning strategy a tester should subdivide the input domain into "small numbers" of subdomains, which are disjoint and collectively constitute a partition of $\mathscr{D}$

Assumption

Equivalence classes are built assuming that the program under test exhibits the same behaviour on all elements of the same subset. One element for each subset is selected to form $\mathscr{T}$

Results?

Quality of $\mathscr{T}$ depends from experience, familiarity with requirements, access and familiarity with the code

Equivalence partitioning

How to ...

using the equivalence partitioning strategy a tester should subdivide the input domain into "small numbers" of subdomains, which are disjoint and collectively constitute a partition of $\mathscr{D}$

Assumption

Equivalence classes are built assuming that the program under test exhibits the same behaviour on all elements of the same subset. One element for each subset is selected to form $\mathscr{T}$

Results?

Quality of $\mathscr{T}$ depends from experience, familiarity with requirements, access and familiarity with the code

Equivalence partitioning

How to ...

using the equivalence partitioning strategy a tester should subdivide the input domain into "small numbers" of subdomains, which are disjoint and collectively constitute a partition of $\mathscr{D}$

Assumption

Equivalence classes are built assuming that the program under test exhibits the same behaviour on all elements of the same subset. One element for each subset is selected to form $\mathscr{T}$

Results?

Quality of $\mathscr{T}$ depends from experience, familiarity with requirements, access and familiarity with the code

Faults targeted

Simple partitioning:

- set of legal (valid) and not legal input
- try to identify requirements explicitely referring to different sets
  - Consider an application related to the management of pensions and retirements – (Req1:$age \in [1, .., 15]$, Req2:$age \in [16, .., 65]$, Req3:$age \in [16, .., 70]$, Req2.2:$age \in [65, .., 120]$ and Req3.2:$age \in [70, .., 120]$)
  - $gender \in \{M, F\}$
- above and below

Formalizing the approach

Relations helping a tester in partitioning are of the form:

$$R \subset \mathscr{I} \times \mathscr{I}$$

where the relation is reflexive, transitive and symmetric

Given a single requirement we can define a function:

$$R_i : \mathscr{I} \to \mathscr{I}$$

The grocery (simple one)

Consider a method `getPrice` that takes the name of a grocery item consults a database of prices and return the unit price for the item. How would you partion the input?

$$pFound : \mathscr{I} \to \mathscr{I}$$

Elements in the database *pF* and elements not in the database *pNF*. They constitute a partion of $\mathscr{I}$

Formalizing the approach

Relations helping a tester in partitioning are of the form:

$$R \subset \mathscr{I} \times \mathscr{I}$$

where the relation is reflexive, transitive and symmetric
Given a single requirement we can define a function:

$$R_i : \mathscr{I} \rightarrow \mathscr{I}$$

The grocery (simple one)

Consider a method `getPrice` that takes the name of a grocery item consults a database of prices and return the unit price for the item.
How would you partion the input?

$$pFound : \mathscr{I} \rightarrow \mathscr{I}$$

Elements in the database *pF* and elements not in the database *pNF*.
They constitute a partion of $\mathscr{I}$

Formalizing the approach

Relations helping a tester in partitioning are of the form:

$$R \subset \mathscr{I} \times \mathscr{I}$$

where the relation is reflexive, transitive and symmetric
Given a single requirement we can define a function:

$$R_i : \mathscr{I} \to \mathscr{I}$$

The grocery (simple one)

Consider a method `getPrice` that takes the name of a grocery item consults a database of prices and return the unit price for the item.
How would you partion the input?

$$pFound : \mathscr{I} \to \mathscr{I}$$

Elements in the database *pF* and elements not in the database *pNF*.
They constitute a partion of $\mathscr{I}$

Formalizing the approach

Relations helping a tester in partitioning are of the form:

$$R \subset \mathscr{I} \times \mathscr{I}$$

where the relation is reflexive, transitive and symmetric
Given a single requirement we can define a function:

$$R_i : \mathscr{I} \to \mathscr{I}$$

The grocery (simple one)

Consider a method `getPrice` that takes the name of a grocery item consults a database of prices and return the unit price for the item.
How would you partion the input?

$$pFound : \mathscr{I} \to \mathscr{I}$$

Elements in the database *pF* and elements not in the database *pNF*.
They constitute a partion of $\mathscr{I}$

Examples

Printers

Consider an automatic printer testing application named `pTest`. The application takes the manufacturer name and the model of a printer as input and selects a test script from a list. The script is then executed to test the printer. Our goal is to test if the script selection part of the application is implemented correctly. Different types of printers available (Inkjet, laserjet, multifunction) and (color or B/W).

How would you partion the input?

It is possible to have overlapping among equivalence classes defined by different requirements

Words Count I

The `wordCount` method takes a word *w* and a file name *f* and returns the number of occurrences of *w* in the text contained in the file. An exception is raised if there is no file with name *f*.

How would you partion the input?

Examples

Printers

Consider an automatic printer testing application named `pTest`. The application takes the manufacturer name and the model of a printer as input and selects a test script from a list. The script is then executed to test the printer. Our goal is to test if the script selection part of the application is implemented correctly. Different types of printers available (Inkjet, laserjet, multifunction) and (color or B/W).

How would you partion the input?

It is possible to have overlapping among equivalence classes defined by different requirements

Words Count I

The `wordCount` method takes a word *w* and a file name *f* and returns the number of occurrences of *w* in the text contained in the file. An exception is raised if there is no file with name *f*.

How would you partion the input?

Examples

Printers

Consider an automatic printer testing application named `pTest`. The application takes the manufacturer name and the model of a printer as input and selects a test script from a list. The script is then executed to test the printer. Our goal is to test if the script selection part of the application is implemented correctly. Different types of printers available (Inkjet, laserjet, multifunction) and (color or B/W).

How would you partion the input?

It is possible to have overlapping among equivalence classes defined by different requirements

Words Count I

The `wordCount` method takes a word *w* and a file name *f* and returns the number of occurrences of *w* in the text contained in the file. An exception is raised if there is no file with name *f*.

How would you partion the input?

Examples

Words Count II

Now suppose to have access to the code of wordCount:

```
begin
  string w,f;
  input (w,f);
  if (!exists(f)) {raise exception; return(0)};
  if (length(w)==0) return (0);
  if (empty(f)) return (0);
  return (getCount(w,f));
end
```

How would you partion the input, now?

In some cases the equivalence classes are based on the output generated by the program

Examples

Words Count II

Now suppose to have access to the code of wordCount:

```
begin
  string w,f;
  input (w,f);
  if (!exists(f)) {raise exception; return(0)};
  if (length(w)==0) return (0);
  if (empty(f)) return (0);
  return (getCount(w,f));
end
```

How would you partion the input, now?

In some cases the equivalence classes are based on the output generated by the program

Examples

Words Count II

Now suppose to have access to the code of wordCount:

```
begin
  string w,f;
  input (w,f);
  if (!exists(f)) {raise exception; return(0)};
  if (length(w)==0) return (0);
  if (empty(f)) return (0);
  return (getCount(w,f));
end
```

How would you partion the input, now?

In some cases the equivalence classes are based on the output generated by the program

The triangle software

The function `checkTriangle(s1,s2,s3)` takes in input three values representing possible sides of a triangle and it returns the classification of the triangle:

- side based classification:
    - isosceles
    - equilateral
    - scalene
- angle based classification:
    - right
    - equiangular
    - obtuse
    - acute

Provide the test cases for such function according to the domain partitioning method

Equivalence classes for variables

There are some guidelines to define equivalence classes on the base of variables domains and defined requirements. They reflect possible implementation choices related to explicit knowledge or implicit one:

- Range (implicitly or explicitly defined): one class with values inside the range and two with values outside the range
- Strings: at least one containing all legal strings and one containing all illegal strings (lenght or sematic constraints – e.g. password constraints)
- Enumerations: each value in a separate class
- Arrays: one class containing all legal arrays, one the empty array, and one larger than the expected size
- Compound Data Types (e.g. age and name): combine the classes composing the compound type

Unidimensional vs. Multidimensional partitioning

Unidimensional

Each input variable is considered per-se and classes are combined to cover all the possible equivalence classes

Multidimensional

The Cartesian product of equivalence classes is considered and test derived accordingly.

Example

Consider a software taking in input two parameters according to the following constraints:

$$3 \leq x \leq 7, 5 \leq y \leq 9$$

Partitions and cardinality of the test set for the two different approaches?

Unidimensional vs. Multidimensional partitioning

Unidimensional

Each input variable is considered per-se and classes are combined to cover all the possible equivalence classes

Multidimensional

The Cartesian product of equivalence classes is considered and test derived accordingly.

Example

Consider a software taking in input two parameters according to the following constraints:

$$3 \leq x \leq 7, 5 \leq y \leq 9$$

Partitions and cardinality of the test set for the two different approaches?

Partitioning

A Systematic Procedure

- Identify input domains – read requirements carefully, identify input and output variables and their types, as well as conditions related to them.
- Equivalence classing – partition the set of values of each variable into disjoint subsets
- Combine equivalence classes – combine equivalence classes
- Identify infeasible equivalence classes – combination of data that cannot be input to the application under test

The Boiler Control System (BCS)

BCS

The control system takes in input:

- A command: $cmd = (temp|shut|cancel)$
- When *temp* is selected $tempch = -10|-5|5|10$

Input can be provided via a GUI or via a configuration file.
How would you partition the input domain?

BCS input domain

Variable	Type	Value(s)
InputMechanism	Enumerated	$\{GUI, file\}$
FileName	String	*A file name*
cmd	Enumerated	$\{temp, cancel, shut\}$
tempch	Enumerated	$\{-10, -5, 5, 10\}$

The Boiler Control System (BCS)

BCS

The control system takes in input:

- A command: $cmd = (temp|shut|cancel)$
- When *temp* is selected $tempch = -10|-5|5|10$

Input can be provided via a GUI or via a configuration file.
How would you partition the input domain?

BCS input domain

Variable	Type	Value(s)
InputMechanism	Enumerated	$\{GUI, file\}$
FileName	String	*A file name*
cmd	Enumerated	$\{temp, cancel, shut\}$
tempch	Enumerated	$\{-10, -5, 5, 10\}$

The Boiler Control System (BCS)

BCS Equivalence Classes

Variable	Type	Value(s)
InputMechanism	Enumerated	$\{\{GUI\}, \{file\}, \{undefined\}\}$
FileName	String	*f_valid*, *f_invalid*
cmd	Enumerated	$\{\{temp\}, \{cancel\}, \{shut\}, \{c\_invalid\}\}$
tempch	Enumerated	$\{\{-10\}, \{-5\}, \{5\}, \{10\}, \{c\_invalid\}\}$

- Infeasible equivalence classes
- Some considerations on the impact of GUI on testing activities

The Boiler Control System (BCS)

BCS Equivalence Classes

Variable	Type	Value(s)
InputMechanism	Enumerated	$\{\{GUI\}, \{file\}, \{undefined\}\}$
FileName	String	*f_valid*, *f_invalid*
cmd	Enumerated	$\{\{temp\}, \{cancel\}, \{shut\}, \{c\_invalid\}\}$
tempch	Enumerated	$\{\{-10\}, \{-5\}, \{5\}, \{10\}, \{c\_invalid\}\}$

- Infeasible equivalence classes
- Some considerations on the impact of GUI on testing activities

ToC

Boundary-value analysis

Experience indicates that programmers make mistakes in processing values at and near the boundaries of equivalence classes

Boundary-value analysis

test-selection techniques that targets faults in applications at the boundaries of equivalence classes.

Once the input domain has been identified:

- Partition the input domain using unidimensional partitioning
- Identify the boundaries of each partition
- Select test data such that each boundary value occurs in at least one test input

Boundary-value analysis

Experience indicates that programmers make mistakes in processing values at and near the boundaries of equivalence classes

Boundary-value analysis

test-selection techniques that targets faults in applications at the boundaries of equivalence classes.

Once the input domain has been identified:

- Partition the input domain using unidimensional partitioning
- Identify the boundaries of each partition
- Select test data such that each boundary value occurs in at least one test input

Boundary-value analysis

Experience indicates that programmers make mistakes in processing values at and near the boundaries of equivalence classes

Boundary-value analysis

test-selection techniques that targets faults in applications at the boundaries of equivalence classes.

Once the input domain has been identified:

- Partition the input domain using unidimensional partitioning
- Identify the boundaries of each partition
- Select test data such that each boundary value occurs in at least one test input

BA Example

The *findPrice* procedure

Two integer parameter *code* and *quantity* with the following input domain:

- $99 \leq code \leq 999$
- $1 \leq quantity \leq 100$

Which are the relevant partitions?
Which are the relevant boundary values?

BA Example

The *findPrice* procedure

Two integer parameter *code* and *quantity* with the following input domain:

- $99 \leq code \leq 999$
- $1 \leq quantity \leq 100$

Which are the relevant partitions?
Which are the relevant boundary values?

BA Example

Consider the following test set:

$$
\begin{aligned}
T = \{ \quad & t_1 : (code = 98, & quantity = 0), \\
& t_2 : (code = 99, & quantity = 1), \\
& t_3 : (code = 100, & quantity = 2), \\
& t_4 : (code = 998, & quantity = 99), \\
& t_5 : (code = 999, & quantity = 100), \\
& t_6 : (code = 1000, & quantity = 101), \\
\}
\end{aligned}
$$

Minimal but:

```
public void fP(int code, int quantity) {
  if (code < 99 && code > 999)
    {display_error("invalid code"); return;}
    // Validity check for quantity is missing!
    // Begin processing code and quantity
  ...
}
```

BA Example

Consider the following test set:

$$
\begin{aligned}
T = \{ \quad & t_1 : & (code = & \ 98, & quantity = & \ 0), \\
& t_2 : & (code = & \ 99, & quantity = & \ 1), \\
& t_3 : & (code = & \ 100, & quantity = & \ 2), \\
& t_4 : & (code = & \ 998, & quantity = & \ 99), \\
& t_5 : & (code = & \ 999, & quantity = & \ 100), \\
& t_6 : & (code = & \ 1000, & quantity = & \ 101), \\
\}
\end{aligned}
$$

Minimal but:

```
public void fP(int code, int quantity) {
  if (code < 99 && code > 999)
    {display_error("invalid code"); return;}
    // Validity check for quantity is missing!
    // Begin processing code and quantity
  ...
}
```

Boundary Values Analysis Example

textSearch

Consider the method *textsearch* that takes in input a string `s` and a text `txt` and checks if the string is a substring of the text. In such a case it returns in `p` the position of the first character, -1 otherwise.

- For strings empty string and non empty string classes could be the most natural partitions if no other information are available.
- use of partitions from output variables

"Voilà! In view, a humble vaudevillian veteran, cast vicariously as both victim and villain by the vicissitudes of Fate. This visage, no mere veneer of vanity, is it vestige of the vox populi, now vacant, vanished, as the once vital voice of the verisimilitude now venerates what they once vilified ..."

On Combining Boundary Values

Combining Boundary Values

Boundary values for an input set should be tested in isolation avoiding interferences from other input sets

Relationship among input variables

- Relationships amongst the input variables must be examined carefully while identifying boundaries along the input domain.
- Additional tests may be obtained when using a partition of the input domain obtained by taking the product of equivalence classes created using individual variables

ToC

Category Partition Method

CP Method

Mixed manual/automatic approach consisting of eight successive steps based approach to go from requirements to test scripts

- Analyze specification
- Identify Categories
- Partition Categories
- Identify Constraints
- (Re)write test specification
- Process Specification
- Evaluate Generator Output
- Generate Test Scripts

Category Partition Method

the findPrice procedure (2nd version)

`findPrice(code,quantity,weight)`

- `code`: string of eight digits
- `qty`: quantity purchased
- `weight`: weight of the purchased item

The procedure accesses a database to find and display the unit price, the description, and the total price of the item corresponding to `code`.

Leftmost digit	Interpretation
0	Ordinary grocery items such as bread, magazines soup
2	Variable-weight items such as meats, fruits, and vegetables
3	Health-related items such as tylenol, bandaids, and tampons
5	Coupon; digit 2(dollars), 3 and 4 (cents) specify the discounts
1, 6-9	unused

Analyze Specification

The tester identify each functional unit that can be tested separately

E.g. it could be the case that the `findPrice` procedures implements in a single component the functionalities related to the retrieval of information from the database

Identify categories

For each testable unit the specification is analyzed and inputs isolated. Also objects in the environment are considered. Then the relevant characteristics (category) of each parameter are identified

findPrice

Categories:

- *code*: length, leftmost digits, remaining digits
- *quantity*: integer quantity
- *weight*: float quantity
- *database*: contents

Partition Categories

For each category different cases (choices) against which to test the functional units are identified.

- Parameters
  - *code*:
    - Length: Valid (8 digits), Invalid ($<$ or $>$ 8)
    - leftmost digit: 0,2,3,5,others
    - remaining digits: valid string, invalid string
  - *quantity*: valid, invalid
  - *weight*: valid, invalid
- Environments
  - *Database*: item exists, item does not exist

Identify Constraints

- In this step constraints among choices are specified and used to exclude infeasible tests
  - **`if`** property . . .
- A computer readable format permits to automate the activity

(Re)write test specification

The tester write a complete test specification using a Test Specification Language (TSL), and taking into account the information derived in the previous steps

Note: The TSL is suitable for being automatically parsed in the next step

Process Specification

TSL specification are processed to derive test frames (test template).

Note: The strategy has been conceived as a BB approach. Code is not available while you work on defining the tests

Evaluate Generator Output

Generated tests are analyzed for redundancy of missing cases

Note: Identification of errors or missing cases can send back the procedure to a previous step

Generate Test Scripts

Test frames are finally grouped into test scripts

CP is mainly a systematization of the equivalence partitioning and boundary value analysis

Generate Test Scripts

Test frames are finally grouped into test scripts

CP is mainly a systematization of the equivalence partitioning and boundary value analysis

In Summary

Test derivation strategies based on characteristics of the input sets

- Partitioning
- Boundary analysis
- Category Partition