Test Adequacy Criteria and Structural Testing

Daniel Sundmark

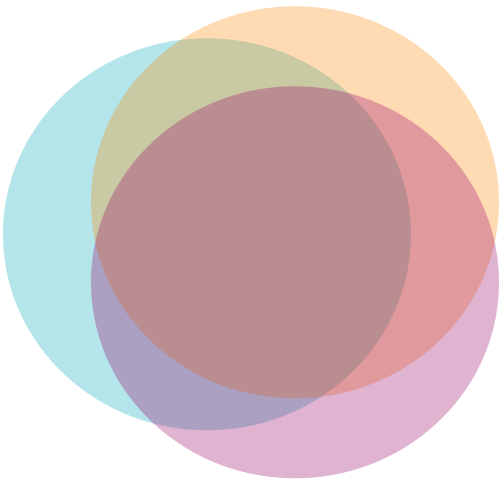
"How do we know what we tested (or did not test)?"

In this Lecture

- Structural Testing Fundamentals
- Adequacy Criteria and Coverage
  - Graph-based Adequacy Criteria
  - Logic-based Adequacy Criteria
- Important Considerations for Structural Testing

Fundamentals of Structural Testing

Fundamentals of Structural Testing



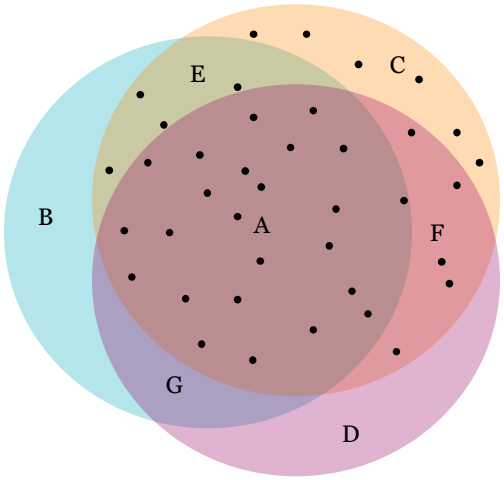
Fundamentals of Structural Testing

Fundamentals of Structural Testing

Fundamentals of Structural Testing

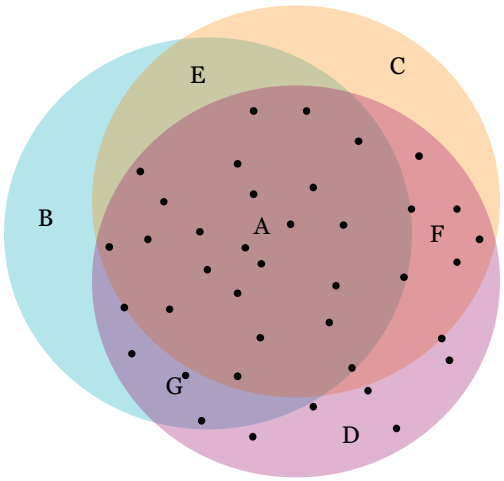
Functional Testing Focus

A: Desired, specified and implemented software behaviour
B: Desired, but not specified or implemented software behaviour
C: Specified, but not desired or implemented software behaviour
D: Implemented, but not desired or specified software behaviour
E: Desired and specified, but not implemented software behaviour
F: Specified and implemented, but not desired software behaviour
G: Desired and implemented, but not specified software behaviour



Structural Testing Focus

A: Desired, specified and implemented software behaviour
B: Desired, but not specified or implemented software behaviour
C: Specified, but not desired or implemented software behaviour
D: Implemented, but not desired or specified software behaviour
E: Desired and specified, but not implemented software behaviour
F: Specified and implemented, but not desired software behaviour
G: Desired and implemented, but not specified software behaviour



Test Design Techniques

- Can be seen as functions that, based on a set of rules, generate test cases from software artifacts (e.g., specifications or code)

Structural Testing

Functional testing is about covering the **specification**

Structural testing is about covering the **implementation**

Structural Testing

Functional testing is about covering the **specification**

Structural testing is about covering the **implementation**

Note that you often need the specification here to know the expected output, though

Structural Test Fundamentals

Adequacy Criteria and Coverage

- How do you know how well you cover an implementation?

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

Structural Test Fundamentals

Adequacy Criteria and Coverage

- How do you know how well you cover an implementation?

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

This is a statement.

Statement coverage is the most commonly used form of code coverage.

Structural Test Fundamentals

Adequacy Criteria and Coverage

- How do you know how well you cover an implementation?

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

This is a statement.

Statement coverage is the most commonly used form of code coverage.

Adequacy criterion

Structural Test Fundamentals

Adequacy Criteria and Coverage

- How do you know how well you cover an implementation?

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

This is a statement.

Statement coverage is the most commonly used form of code coverage.

Coverage item

Structural Test Fundamentals

Adequacy Criteria and Coverage

- *Adequacy criteria*
  - Formal criteria used for stating how well test suites exercise the implementation
  - *Inadequacy criteria*?
  - An adequacy criterion applied to an implementation will result in a set of ***coverage items*** that should be exercised during testing

Structural Test Fundamentals

Adequacy Criteria and Coverage

- *Adequacy criteria*
  - Formal criteria used for stating how well test suites exercise the implementation
  - *Inadequacy criteria*?
  - An adequacy criterion applied to an implementation will result in a set of ***coverage items*** that should be exercised during testing

coverage items are also known as **test items, test obligations, test requirements**, etc...

Testing terminology is a jungle...

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

- Hence, always a number between 0 and 1 (often given as a percentage)

Structural Test Fundamentals

Adequacy Criteria and Coverage

- Given an adequacy criterion *AC*, a set of test cases *T* and a system under test *S*,

$$Coverage_{AC} = \frac{|Tested_{AC}(T,S)|}{|Existing_{AC}(S)|}$$

Example

- The below code has 14 statements

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

Example

Test case
(input, expected output)

- Let's say that 10 of these statements are exercised during testing.

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

Example

- This gives us a statement coverage of 10/14 = 71%

```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```