Design by Contract from Testing Perspective: Binding Theory to Practice

Zoran Horvat

PRINCIPAL CONSULTANT AT CODING HELMET

@zoranh75 csharpmentor.com

Categorization of Exceptions

throw...	catch...
`NetworkException` `SocketException`	Consult configured recovery policies — Connectivity loss wait 1 sec., repeat — Timeout wait 10 sec., repeat
`HttpException`	Consult recovery policies
`IOException`	Consult recovery policies
`ArgumentNullException`	Make the argument non-null, repeat **But how?**
`OutOfMemoryException` `StackOverflowException`	No memory/stack space to recover
`IOException` (from an arithmetic operation)	True exception — No way to predict it — No way to recover and repeat

Categorization of Exceptions

Expected errors	Critical errors	Unexpected errors	Other errors
`HttpException`	`OutOfMemoryException`	`IOException` from a non-IO-bound operation	`ArgumentException`
`NetworkException`	`StackOverflowException`		`ArgumentNullException`
`IOException`			`NullReferenceException`
`SocketException`			`ObjectDisposedException`
			`InvalidOperationException`
			`IndexOutOfRangeException`
Use recovery policy to handle error Proceed as if nothing has happened	Don't try to handle — Exit the application — Maybe try to save (but don't expect too much)	No policy to handle — Abandon operation — GUI pop-up — HTTP 500 Internal Error — HTTP 503 Temporarily Unavailable, etc.	These are indicative of a **bug** — Don't try again — Log all the details — Fix the bug in code

No way to recover
from negative input

```
double SquareRoot(double x)
{
    return Math.Sqrt(x);
}
```

The function
can only fail

Don't cross this line either

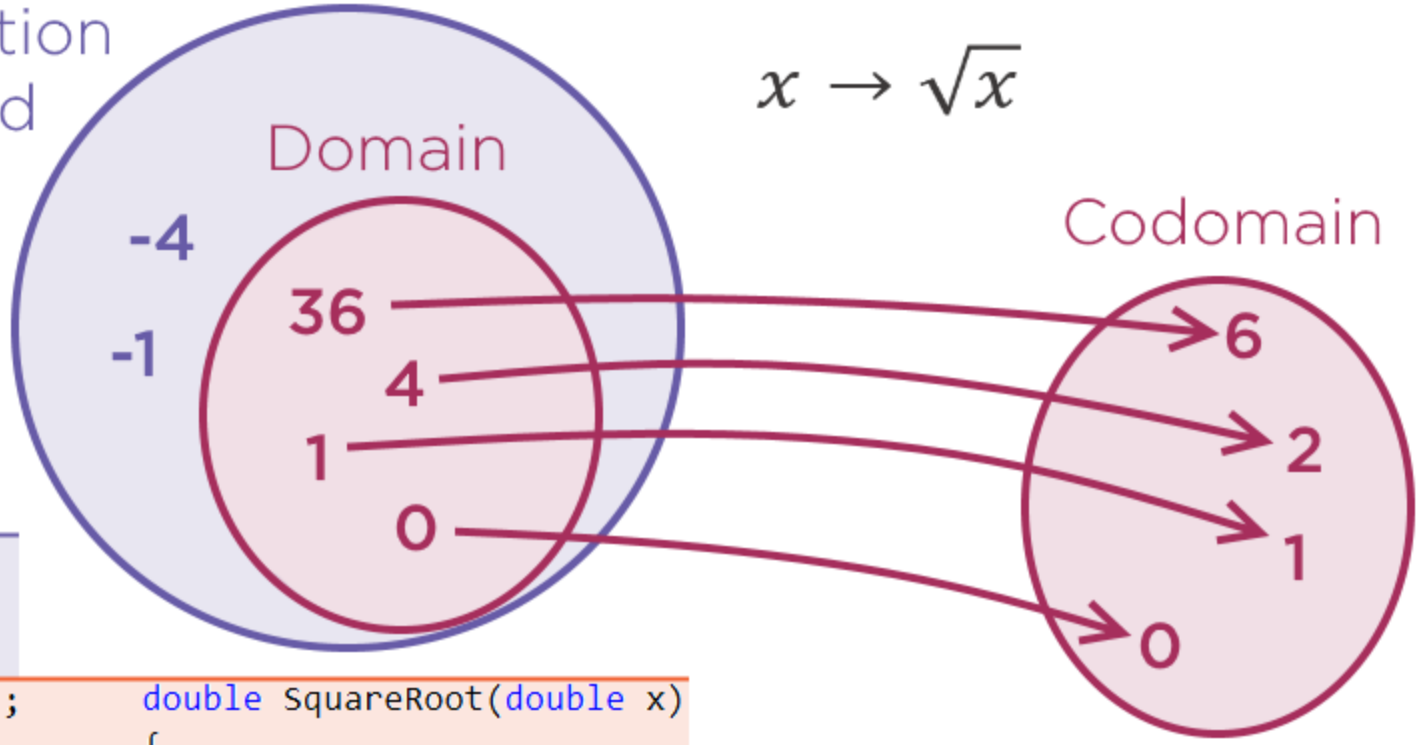
Don't cross this line with $x < 0$

```
void OuterCaller()
{
    ...
    Calculate();
    ...
}
```

```
double Calculate()
{
    double x = data.Load();
    double y = SquareRoot(x);
    return 2 * y + x;
}
```

```
double SquareRoot(double x)
{
    return Math.Sqrt(x);
}
```

Mathematical function
is not defined
outside of its domain



$$x \rightarrow \sqrt{x}$$

```
void OuterCaller()
{
    ...
    Calculate();
    ...
}
```

```
double Calculate()
{
    double x = data.Load();
    double y = SquareRoot(x);
    return 2 * y + x;
}
```

```
double SquareRoot(double x)
{
    return Math.Sqrt(x);
}
```

Method is not defined for negative input

Design by Contract

- Calling method and called method establish a contract

Preconditions

- Boolean condition which must be satisfied *before* a method is invoked

Postcondition

- Boolean condition which must be satisfied *after* a method completes

If you want to improve your software, don’t just test more; develop better.

Steve McConnell, *Code Complete*

Summary

Introduced Design by Contract (DbC)

- Preconditions and postconditions
- Adhering to Liskov Substitution Principle (LSP)
- DbC & LSP are the winning combination

Unit testing in presence of contracts

- Avoid unit tests already covered by contract assertions
- Assertions are checked every time a method executes
- Moves focus to system-level tests

Summary

Configuring contracts

- Heavy checks in test environment
- Strip it off for fastest production code
- Example test suite reduced by 30%

Bertrand Meyer,
Object-Oriented Software Construction

Course Summary

Bird's eye view on making tests easier to maintain

- Make better production code
- Make good tests for the good code

Techniques to improve code

- Make friends with Abstract Data Types
- Make friends with Design by Contract

Techniques to improve tests

- No code duplication
  - Use class inheritance
  - Use object composition
- Each testing class to do one thing

Course Summary

When writing tests

- Use the same set of skills as in production code and that will do it