Contract-based Software Development

Rasmus Guldborg Pedersen

January 2015

Overview

Separating specification and implementation in Code Contracts

How can we separate specification and implementation using Code Contract?

Code Contracts (.NET tool)

Express preconditions, postconditions and object invariants for:

Code Contracts (.NET tool)

Express preconditions, postconditions and object invariants for:

- Static analysis

Code Contracts (.NET tool)

Express preconditions, postconditions and object invariants for:

- Static analysis
- Documentation

Code Contracts (.NET tool)

Express preconditions, postconditions and object invariants for:

- Static analysis
- Documentation
- Runtime checking

Interface

```
public interface ISimpleQueue {
    void Enqueue(object item);
    object Dequeue();
    object ElementAt(int index);
    int Count();
}
```

Contract

```
abstract class ISimpleQueueContract {
    public void Enqueue(object item) {
        Contract.Requires(item != null);
        Contract.Ensures(Count ==
            Contract.OldValue(Count()) + 1);
        Contract.Ensures(ElementAt(Count()) == item);
        // ...
    }
    // ...
}
```

Associating Interface with Contract

```
[ContractClass(typeof(ISimpleQueueContract))]
public interface ISimpleQueue { /* ... */ }

[ContractClassFor(typeof(ISimpleQueue))]
abstract class ISimpleQueueContract { /* ... */ }
```

Inheritance of Preconditions

Subtypes can <u>not</u> have stronger preconditions.

Command Pattern

```
[ContractClass(typeof(ICommandContract))]
public interface ICommand {
    IStack<int> Execute(IStack<int> stack);
}

[ContractClassFor(typeof(ICommand))]
abstract class ICommandContract {
    [Pure]
    IStack<int> Execute(IStack<int> stack) {
        Contract.Requires(stack != null);
        Contract.Ensures(stack != null);
    }
}
```

Stronger Precondition

```
public class PopCommand : ICommand {
    public IStack<int> Execute(IStack<int> stack) {
        Contract.Requires(stack.Count > 0); // Stronger!
        // ...
    }
}
```

Solution

```
public interface ICommand {
    IStack<int> Execute(IStack<int> stack);
    [Pure]
    bool CanExecute(IStack<int> stack);
}
```

The End

"Testing shows the presence, not the absence of bugs."
— Edsger W. Dijkstra