EDUMENT

Code Coverage & Continuous Testing

Test Coverage Analysis

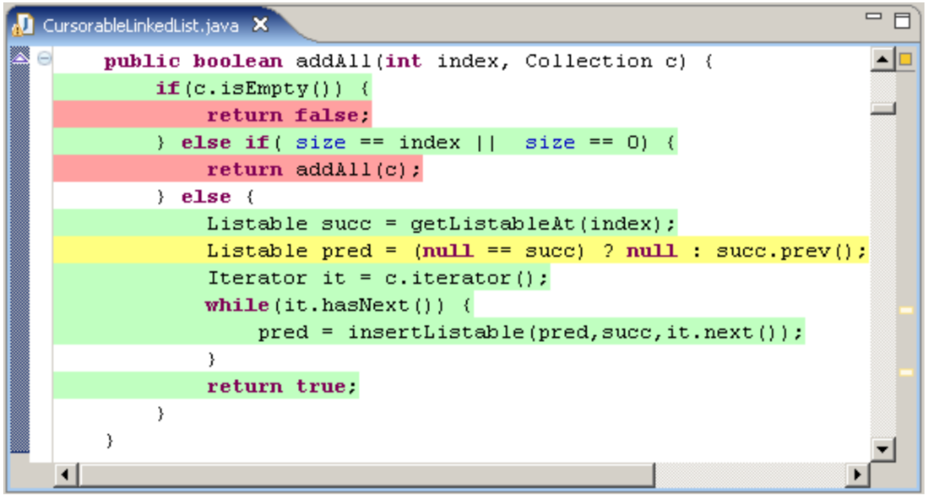
Doing frequent-ish **test coverage** measurements can be highly useful.

Code coverage measures how many lines of code are executed as part of a test run.

🟩 the line was covered

🟨 the line was partially covered

🟥 the line was not covered



```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```

Code Coverage tools

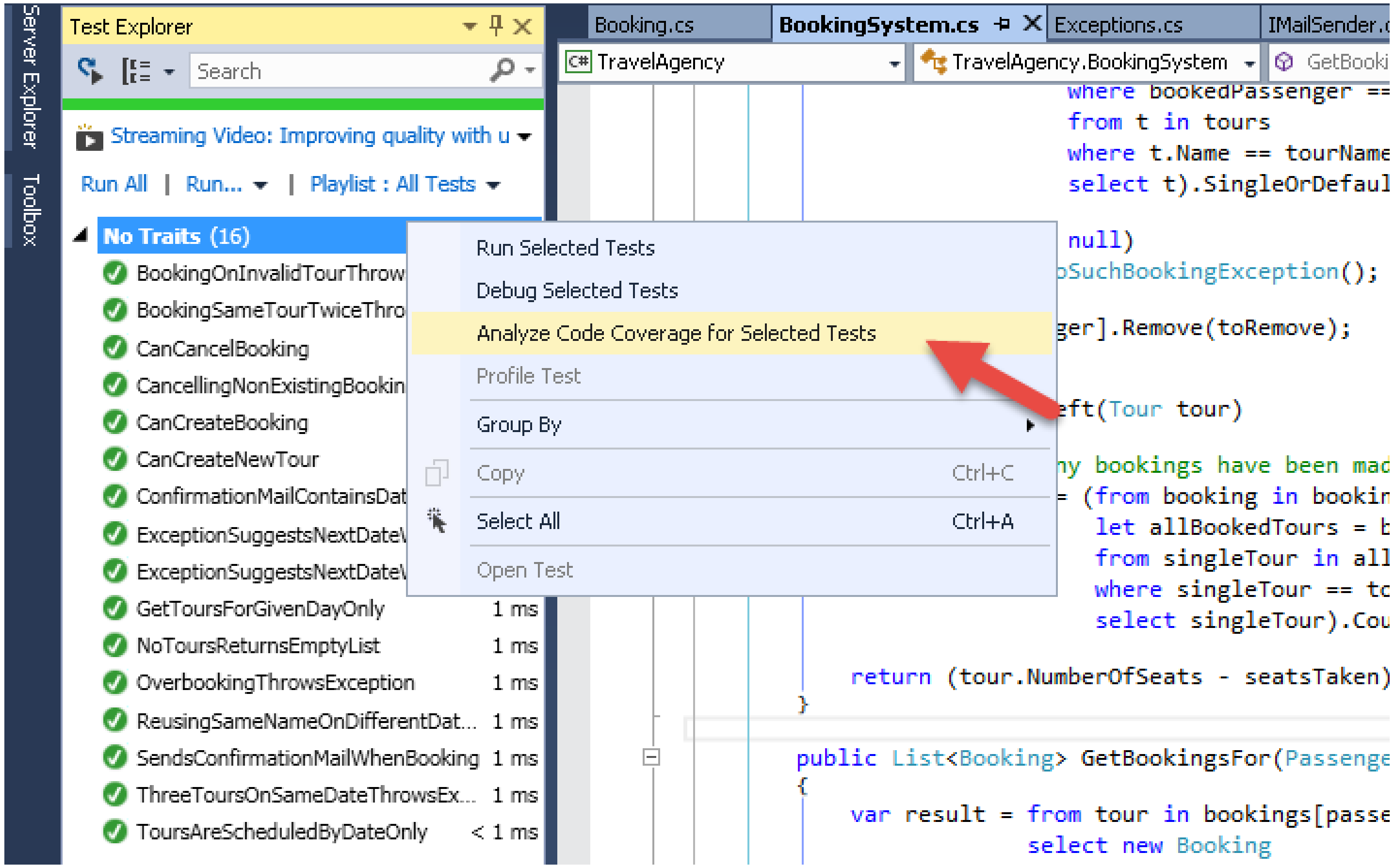
There are many tools available for measuring code coverage

Name	Description
Visual Studio	The build in testing tool in includes a code coverage feature.
Ncover	Popular commercial tool http://www.ncover.com
OpenCover	Open source tool https://github.com/OpenCover/opencover
DotCover	Commercial and free tool from JetBrains https://www.jetbrains.com/dotcover/
Ncrunch	Continuous testing tool http://www.ncrunch.net/
NDepend	Code analysing tool that can import coverage data from the tools above. http://www.ndepend.com

Test Coverage Analysis

EDUMENT

To run the code coverage tests we right click on the test group or individual test:



No Traits (16)

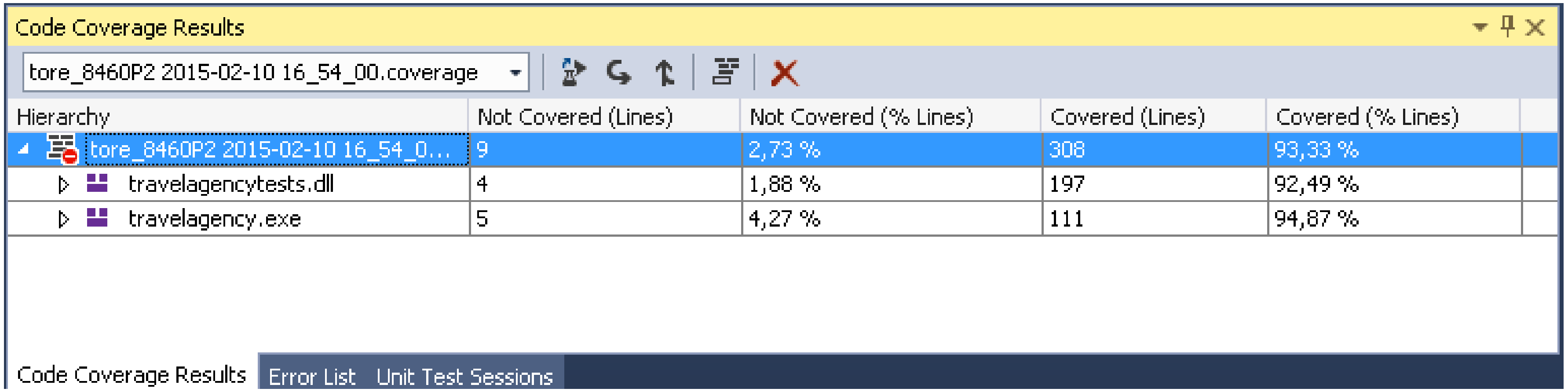
BookingOnInvalidTourThrow	
BookingSameTourTwiceThro	
CanCancelBooking	
CancellingNonExistingBookin	
CanCreateBooking	
CanCreateNewTour	
ConfirmationMailContainsDat	
ExceptionSuggestsNextDate\	
ExceptionSuggestsNextDate\	
GetToursForGivenDayOnly	1 ms
NoToursReturnsEmptyList	1 ms
OverbookingThrowsException	1 ms
ReusingSameNameOnDifferentDat...	1 ms
SendsConfirmationMailWhenBooking	1 ms
ThreeToursOnSameDateThrowsEx...	1 ms
ToursAreScheduledByDateOnly	< 1 ms

```
                              where bookedPassenger ==
                              from t in tours
                              where t.Name == tourName
                              select t).SingleOrDefaul

                              null)
                             oSuchBookingException();

                             ger].Remove(toRemove);


                             eft(Tour tour)

                             ny bookings have been mad
                             = (from booking in bookin
                               let allBookedTours = b
                               from singleTour in all
                               where singleTour == to
                               select singleTour).Cou

        return (tour.NumberOfSeats - seatsTaken)
    }

    public List<Booking> GetBookingsFor(Passenge
    {
        var result = from tour in bookings[passe
                     select new Booking
```

Test Coverage Analysis

EDUMENT

The Code Coverage Results window shows the result:



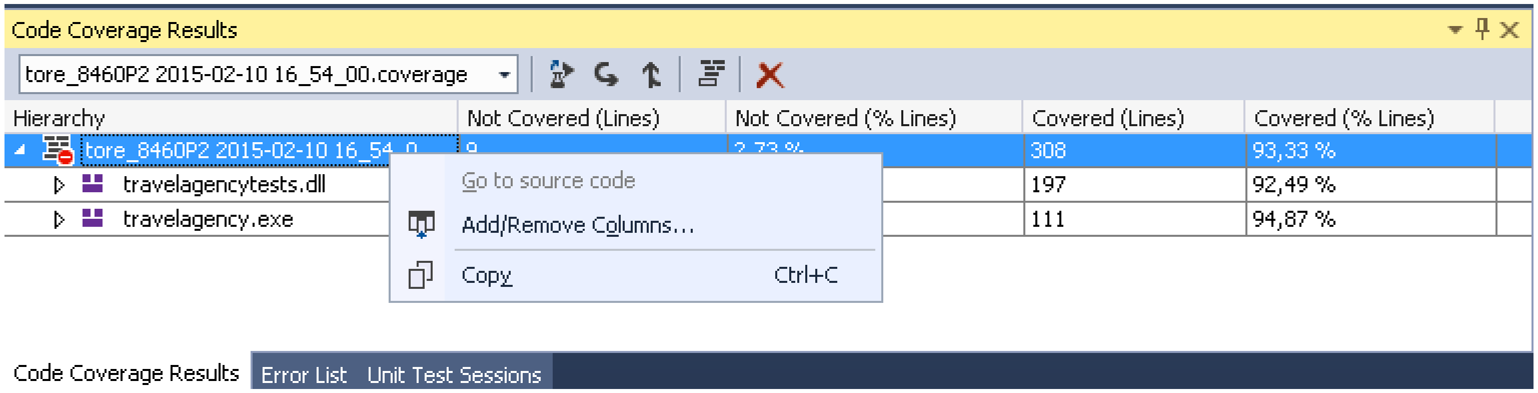
Code Coverage Results

tore_8460P2 2015-02-10 16_54_00.coverage

Hierarchy	Not Covered (Lines)	Not Covered (% Lines)	Covered (Lines)	Covered (% Lines)
tore_8460P2 2015-02-10 16_54_0...	9	2,73 %	308	93,33 %
travelagencytests.dll	4	1,88 %	197	92,49 %
travelagency.exe	5	4,27 %	111	94,87 %

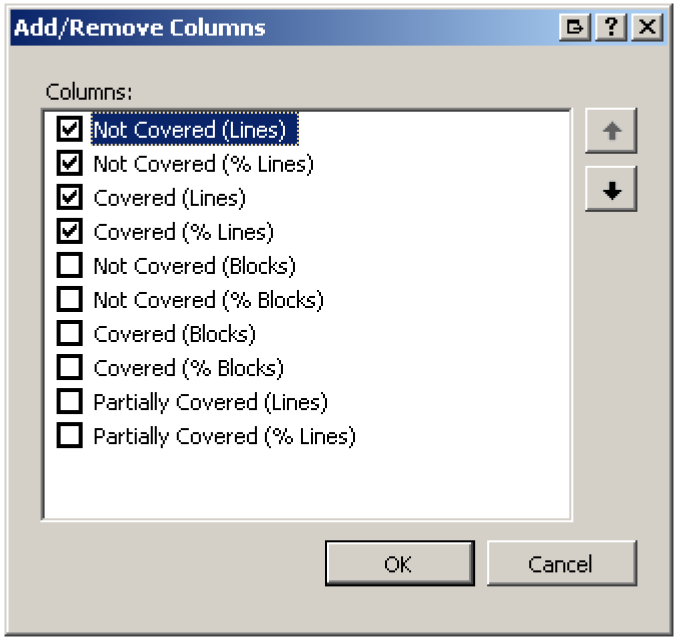
Test Coverage Analysis

Right clicking in the grid allows you to choose what columns to display


Code Coverage Results

tore_8460P2 2015-02-10 16_54_00.coverage

Hierarchy	Not Covered (Lines)	Not Covered (% Lines)	Covered (Lines)	Covered (% Lines)
tore_8460P2 2015-02-10 16_54_0	9	2,73 %	308	93,33 %
travelagencytests.dll			197	92,49 %
travelagency.exe			111	94,87 %


Add/Remove Columns

Columns:

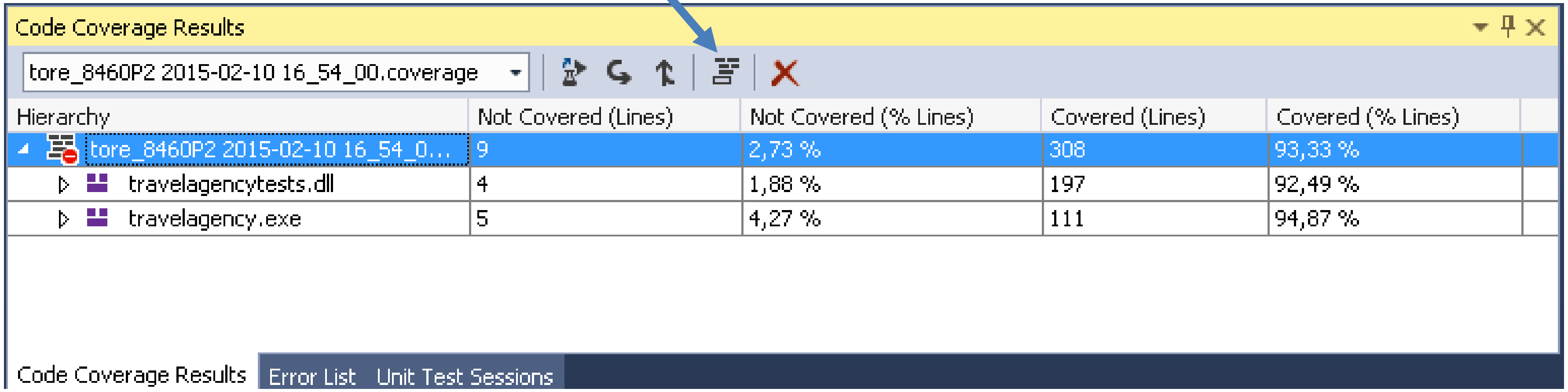
- [x] Not Covered (Lines)
- [x] Not Covered (% Lines)
- [x] Covered (Lines)
- [x] Covered (% Lines)
- [ ] Not Covered (Blocks)
- [ ] Not Covered (% Blocks)
- [ ] Covered (Blocks)
- [ ] Covered (% Blocks)
- [ ] Partially Covered (Lines)
- [ ] Partially Covered (% Lines)

- A block contains a one or more consecutive lines of tested code.
- Showing blocks is the default measurement.

Test Coverage Analysis

Clicking on the “Show Code Coverage Coloring” icon will highlight the code coverage in the source code:

Show Code Coverage Coloring



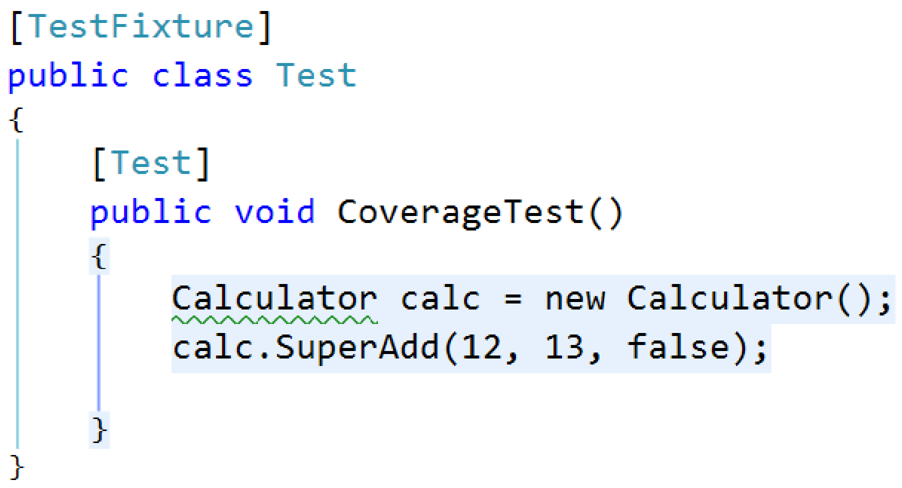
Code Coverage Results

tore_8460P2 2015-02-10 16_54_00.coverage

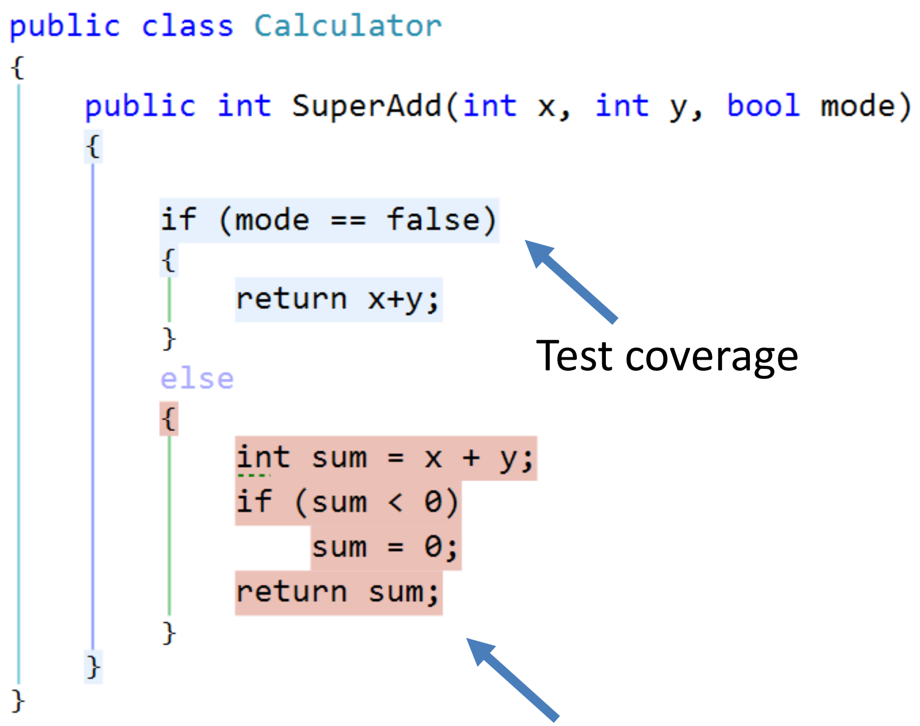
Hierarchy	Not Covered (Lines)	Not Covered (% Lines)	Covered (Lines)	Covered (% Lines)
tore_8460P2 2015-02-10 16_54_0...	9	2,73 %	308	93,33 %
travelagencytests.dll	4	1,88 %	197	92,49 %
travelagency.exe	5	4,27 %	111	94,87 %

Test Coverage Analysis

Sample result



```
[TestFixture]
public class Test
{
    [Test]
    public void CoverageTest()
    {
        Calculator calc = new Calculator();
        calc.SuperAdd(12, 13, false);

    }
}
```



```
public class Calculator
{
    public int SuperAdd(int x, int y, bool mode)
    {

        if (mode == false)
        {
            return x+y;
        }
        else
        {
            int sum = x + y;
            if (sum < 0)
                sum = 0;
            return sum;
        }
    }
}
```

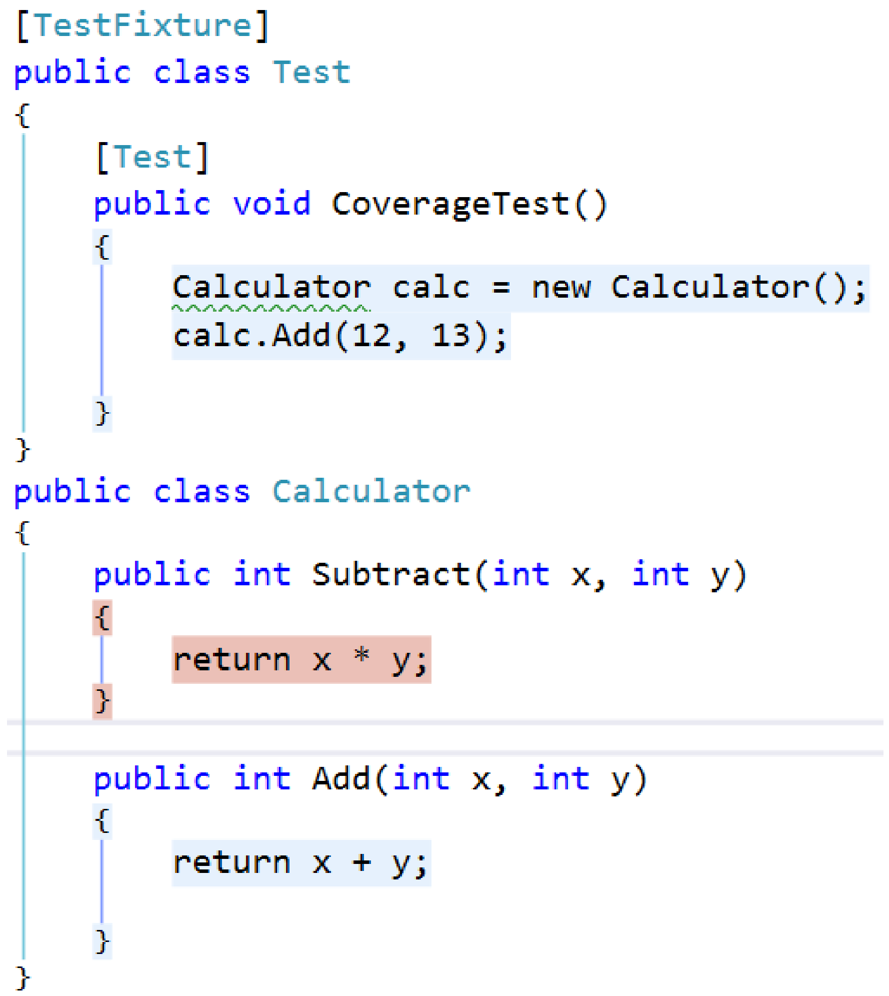
Test coverage

No coverage

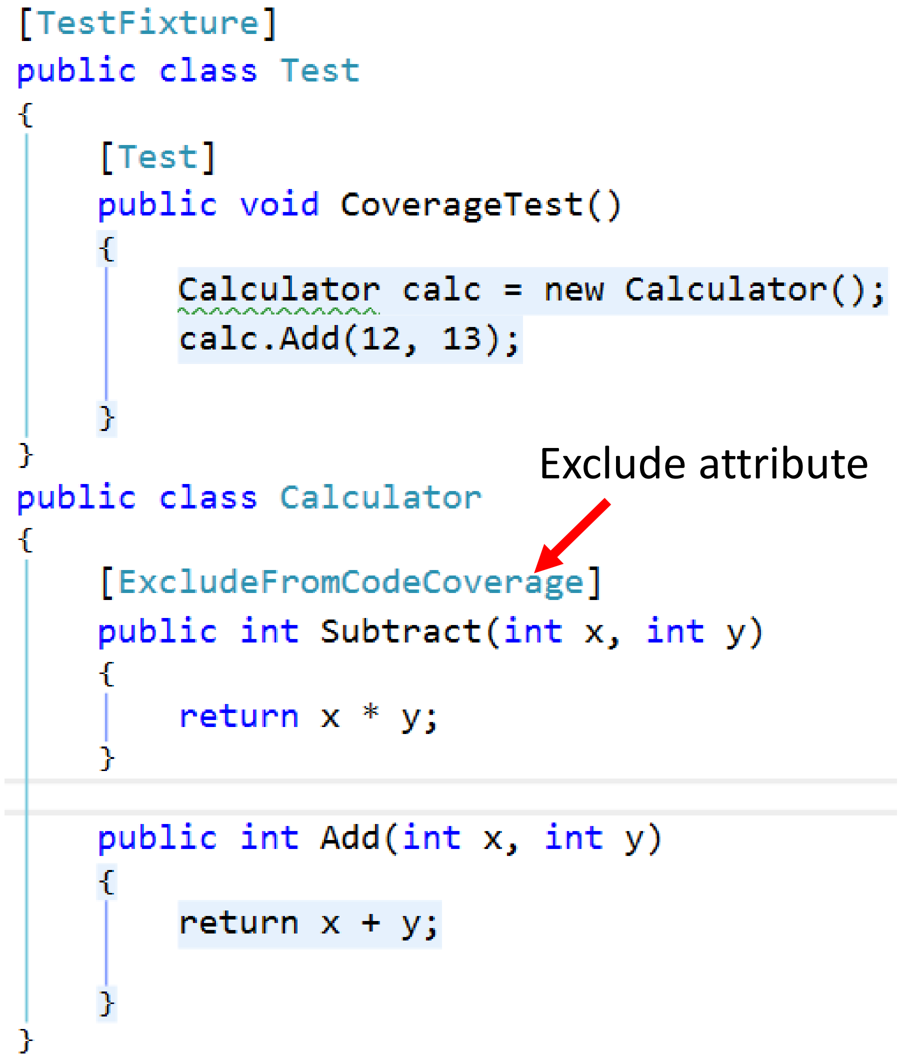
Test Coverage Analysis

EDUMENT

We can ignore code using `[ExcludeFromCodeCoverage]`



```
[TestFixture]
public class Test
{
    [Test]
    public void CoverageTest()
    {
        Calculator calc = new Calculator();
        calc.Add(12, 13);

    }
}
public class Calculator
{
    public int Subtract(int x, int y)
    {
        return x * y;
    }

    public int Add(int x, int y)
    {
        return x + y;

    }
}
```



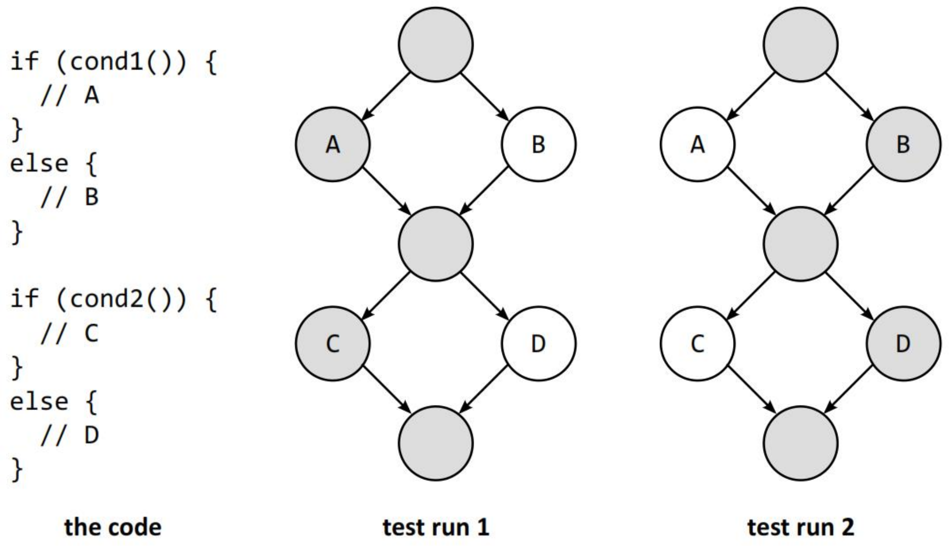
```
[TestFixture]
public class Test
{
    [Test]
    public void CoverageTest()
    {
        Calculator calc = new Calculator();
        calc.Add(12, 13);

    }
}
public class Calculator
{
    [ExcludeFromCodeCoverage]
    public int Subtract(int x, int y)
    {
        return x * y;
    }

    public int Add(int x, int y)
    {
        return x + y;

    }
}
```

Not covered 3 lines,
Coverage 70%

Not covered 0 lines,
Coverage 100%

Test Coverage Analysis

Here's why percentages don't matter so much. "100% coverage" doesn't mean that you test everything, it just means that you run all statements once.



```
if (cond1()) {
  // A
}
else {
  // B
}

if (cond2()) {
  // C
}
else {
  // D
}
```

Not only that, there's diminishing returns in trying to cover everything. (Getter methods, for example. Who has the time?)

Prioritize. Focus on giving the important code good coverage.

Important

Code coverage summary

- Just keep in mind **what** about it is useful.
- Knowing what parts of the business-critical code needs better coverage is useful.
- Reducing the coverage to a single percentage is not so useful.
- Chasing a certain percentage is silly
- 100% coverage is not a goal
- You can still write crappy code with 100% coverage

Resources

- The Way of Testivus –
  Unit Testing Wisdom From An Ancient Software Start-up
  http://www.artima.com/weblogs/viewpost.jsp?thread=203994

- How Much Unit Test Coverage Do You Need? - The Testivus Answer
  http://www.artima.com/forums/flat.jsp?forum=106&thread=204677

- Code Coverage Instrumentation (How the blocks are counted)
  http://blogs.msdn.com/b/phuene/archive/2007/05/03/code-coverage-instrumentation.aspx

- What is a reasonable code coverage % for unit tests (and why)?
  http://stackoverflow.com/questions/90002

EDUMENT

Continuous Testing

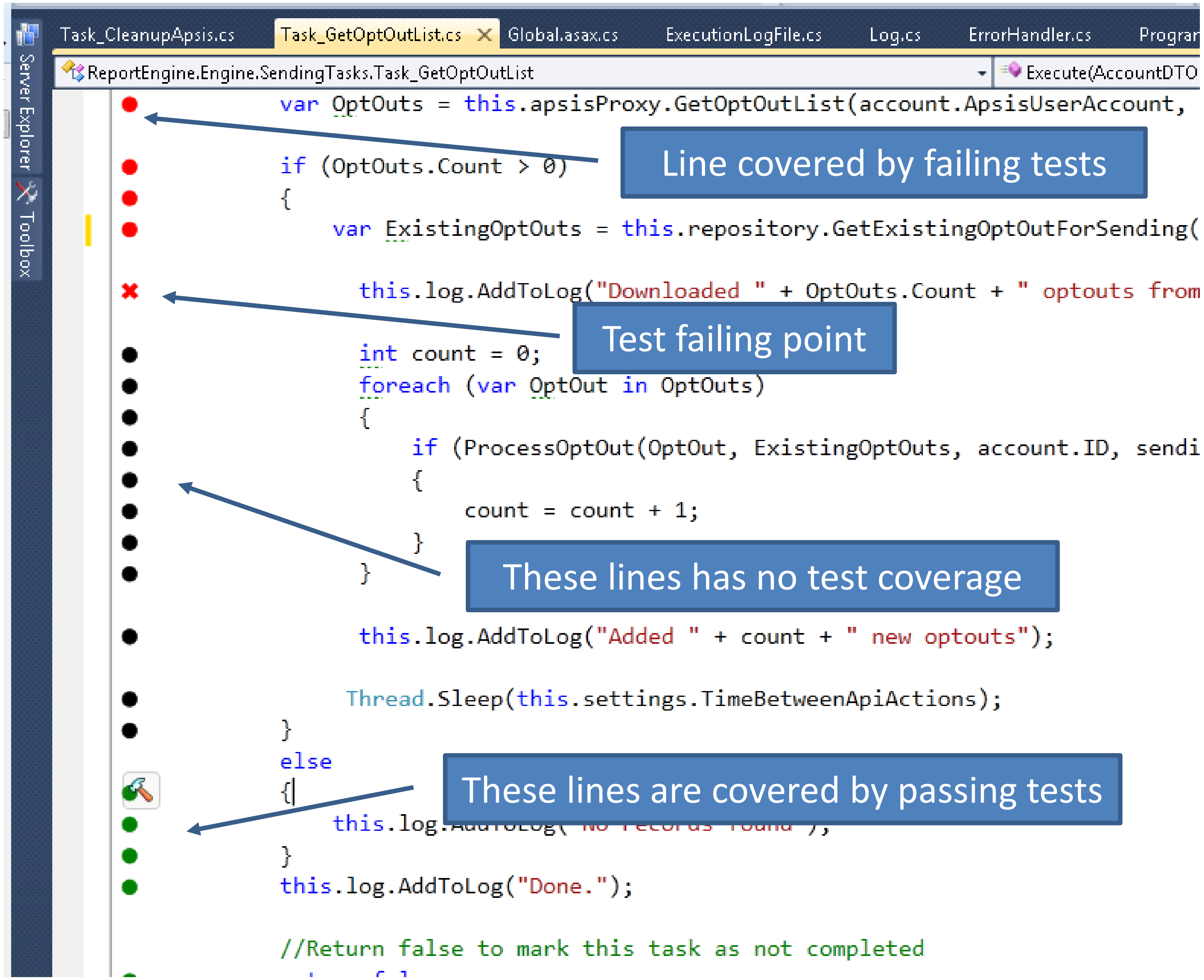
Continuous Testing

Unit tests can also be executed in near real-time inside visual studio. This is called ***Continuous Testing***.

There are a few tools available to support this:

1. **Ncrunch**
   http://www.ncrunch.net/
2. **Continuous Tests**
   http://continuoustests.com/

Continuous Testing



```
●       var OptOuts = this.apsisProxy.GetOptOutList(account.ApsisUserAccount,

●       if (OptOuts.Count > 0)
●       {
●           var ExistingOptOuts = this.repository.GetExistingOptOutForSending(

✖             this.log.AddToLog("Downloaded " + OptOuts.Count + " optouts from

●             int count = 0;
●             foreach (var OptOut in OptOuts)
●             {
●                 if (ProcessOptOut(OptOut, ExistingOptOuts, account.ID, sendi
●                 {
●                     count = count + 1;
●                 }
●             }

●             this.log.AddToLog("Added " + count + " new optouts");

●            Thread.Sleep(this.settings.TimeBetweenApiActions);
●       }
        else
        {
●           this.log.AddToLog(
●       }
●       this.log.AddToLog("Done.");

        //Return false to mark this task as not completed
```

Continuous Testing

Continuous Testing gives improves your TDD efficiency because

- The time from writing a test to failing code is faster, no need to build and run the tests manually.
- You can faster find the lines that is causing trouble.
- You get visual code coverage statistics for free

Visual Studio feels slower?
These tools have several options that you can tweak to optimize the background testing behavior.