Pex - automatisches white box testen mit .net

Thomas Reinwart



Ein Teil der Sicherstellung des Qualitätslevels bei der Entwicklung von Programmen wird mit Unit Tests bei der Ent-

wicklung selber abgedeckt. Mit Unit Tests lässt sich das Verhalten von Methoden in Komponenten testen. Das passiert entweder parallel zur Entwicklung (XP - Extreme programming und TDD test driven development) oder im Nachhinein vom Ersteller des zu testenden Codes. Der Test selber ist eine parameterlose Methode, der die zu testende Methode (kann Übergabeparameter haben) aufruft, das Ergebnis bzw. Verhalten der Methode mit Assert auswertet. (Einem Vergleich von erwartetem Ergebnis und dem Testergebnis) ///A test for ConvertStrings
///</summary>
[TestMethod()]
public void ConvertStringsTest()
{
 string one = string.Empty; // TODO: Initialize to an appropriate value
 string two = string.Empty; // TODO: Initialize to an appropriate value
 string expected = string.Empty; // TODO: Initialize to an appropriate value
 string actual;
 actual = Tools.ConvertStrings(one, two);
 Assert.AreEqual(expected, actual);
 Assert.Inconclusive("Verify the correctness of this test method.");
}
Beispiel eines erzeugten Unit Test Gerüsts

Folgendes Problem ergibt sich aber: Wie stelle ich sicher, dass ausreichend und vor allem sinnvolle Tests erstellt werden? Mittels *Code Coverage* (kann mit externen Tools oder im Visual Studio selber ermittelt werden) lässt sich die Abdeckung, für welche Teile des Sourcecodes bereits Unit Tests vorhanden sind, feststellen. Ein Code Coverage Wert von 80% ist akzeptabel, 100% sind optimal.

Aufgrund von komplexen Zusammenhängen ist es oft nicht einfach, alle notwendigen Varianten der Tests zu erkennen. Zudem ist es aufwendig, alle möglichen Varianten der Übergabeparameter der zu testenden Methode im Unit Test zu implementieren. Vom Entwickler wurden meist eine oder zwei Tests erstellt, die er für den Zeitpunkt der Implementierung für sinnvoll erachtet hat. (Dass er keine Lust hatte weitere Tests zu schreiben schließe ich jetzt mal aus).

Bei einem Projekt schaut das dann so aus: Da der erstellte Code durch die Codecoverage einen positiven Abdeckungsgrad erhalten hat, wurde das Modul von keinem beanstandet und die Komponente released.

Monate später ergibt sich in diesem positiv unit getesteten Modul ein Fehler. Wie konnte das passieren? Genau dieser aufgetretene Fall der übergebenen Parameter wurde bei der Testerstellung einfach nicht bedacht und nie aufgerufen. Ganz nach dem XP Ansatz wird nun der Unittest erstellt, mit dem nun diesen Fehler reproduziert wird. Anschießend wird der Code gefixt, bis der erstellte Unit Test ein positives (grünes) Ergebnis zeigt.

D.h. ich benötige von Anfang an sinnvolle Tests, die alle Varianten abdecken. Dafür gibt es eine Möglichkeit.

1. Pex – automatisches white box testen mit .net

PEX steht für "*Programming EXplorations*" und ist ein Projekt von Microsoft Research. Es ermöglicht die automatische Generierung von Whitebox Unit Tests. PEX besteht aus einem GUI, das in Visual Studio integriert ist. Ebenso können Command Line Befehle genutzt werden.

Eine sehr ausführliche Dokumentation, Tuturial und Samples gibt es online auf der Homepage des Herstellers.

Ein "Whitebox Test" generiert Test Cases indem es den Program Code analysiert. Es sucht dabei nach verschiedenen Fehlerquellen. Als Übergabeparameter an die zu testende Methode werden auch jene Werte übergeben, an die man mit Sicherheit nicht gedacht hat. PEX sucht systematisch nach möglichen Fehlern, und erstellt Tests dafür. Das Ergebnis sind relativ wenige aber dafür sinnvolle Unit Tests.

1.1 Bisherige Möglichkeiten Unit Testzu erstellen

Wie wurden die Unit Tests bisher erstellt:

Visual Studio 2003/2005 und externe Unit Test Tools (z.B. Nunit): alle Test wurden manuell selber codiert. Zur Ermittlung der Unit Test Abdeckung wurde z.B. Ncover benutzt.

Visual Studio 2008: Schon besser - hiermit lässt sich das Gerüst für einen einzelnen Unit Tests erstellen, indem man auf der zu testenden Methode im Contextmenü "Generate Unit Test" klickt. Die Test Abdeckung ist im Studio zu erkennen.



Test Klasse ins Projekt hinzufügen: Context Menü "Create Unit Test" public class Tools

| | pub | lic stat | tic string ConvertS | trin | as(string one. st | ring | two) |
|-----|------|----------|---|------|-------------------------|------|-------|
| | { | | No. of Street | \$ | Run Pex Explorations | Ĩ | 2. S. |
| | | string | result = one[5].To | | Pex | • | |
| | | return | result; | | Organize Usings | • | |
| | } | | | 51 | Create Unit Tests | | |
| 3 | | | | | Create Private Accessor | + | |
| 1.2 | XPui | nd Pex | | | | | |

Wie passt nun XP (Exteme programming) und Pex zusammen?

XP definiert nicht, wie die Unit Test zu schreiben sind. Die bisherigen Unit Test Frameworks (OpenSource Nunit bzw. Visual Studio selber) führen die Unit Test bloß aus, es gab bisher nicht die Möglichkeit automatisiert wirksame Tests zu generieren.

1.3 Installation von Pex

• Windows Vista 32Bit / 64Bit

Andere Windows-Versionen (XP, W2K3, W2K8) sollten ebenfalls geeignet sein, wurden aber nicht getestet. An 64-Bit-Systemen werden nur 32-Bit-Prozesse (Wow64) unterstützt.

NET Framework 2.0, 3.0, 3.5, 4.0

PEXInstallations-Voraussetzung



Pex kann nicht in der Standard Edition sondern nur in der Team Edition von Visual Studio installiert werden.

Lizenz

DevLabs Pre-Release

License: Microsoft Pre-Release Software License, Pre-Release License Agreement, Commercial Use Allowed.

Academic Release

Microsoft Research License Agreement, Non-Commercial Use Only.

| 1.4 Funktionsweise | OpenSource NUnit | | | |
|--|---|--|--|--|
| Pex erzeugt von der Anzahl her wenige aber dafür wirksame Unit Tests. Wirksam in der Hinsicht das eine hohe Code coverage abgedeckt wird | Projektreferenz: nunit.framework.dll Namespace: using NUnit.Framework; | | | |
| Zudem kann jeder generierte Test in einen Test Case gespeichert wer- | Attribut Testklasse: [TestFixture] Attribute Testklasse Konstructor: [SetUn] [TestFixtureSetUn] | | | |
| den um diesen debuggen zu können. | Attribute Testklasse DeKonstructor: [Jettpj [TestFixtureTearDown] [TestFixtureTearDown] | | | |
| Pex funrt den zu testenden Code menrfach mit unterschiedlichen Para- metern aus. Je nachdem wie gut der eigene Code ist, kann es nun auch | using System; | | | |
| zu Effekten kommen. Etwa wenn dahinter eine Datenbank oder Filessys- | using System.Collections.Generic; | | | |
| sondern im Test Environment bleiben. | using System.Linq; | | | |
| Code Input/Output Table | using System.lext; using NUnit.Framework: | | | |
| Public state ForingExensions | | | | |
| if (mine ++ (reine) hall) trev me housestillinging(); trev = are finally limit(); are a set finally limit(); are a set finally limit(); b = a set finally limit(); b | namespace PexSample.Nunit.Test { | | | |
| formati (ma o la value) | [TestFixture] | | | |
| 1 (1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(| public class ToolsTest | | | |
| Parameterized Unit Tests Generated Tests | (Test] | | | |
| // opticalizing any string twise returns non-string [resulted: pairs wais depicalized in confidence of the confidence | <pre>public void _1_Test()</pre> | | | |
| ar mos + Bringforeston-Ognitalise(relate); we rous + Bringforeston-Ognitalise(relate); Anon: Areforestone: Anon: Areforestone: Anon | <pre>string result = PexSample.Tools.ConvertStrings</pre> | | | |
| | ("Hello", "world"); | | | |
| Grafik von http://research.microsoft.com/en-us/projects/Pey/ | <pre>Assert.IsNotNull(result, "Result is empty.");</pre> | | | |
| 1 5 Unit Test Frameworks | } | | | |
| Die Methode ConvertStrings aus der Klasse Tools soll getestet werden: | 2. Pex Anwenden | | | |
| namespace PexSample | 2.1.Sample1 | | | |
| { nublic class Tools | Der Unterschied zu den herkömmlichen Unit Tests ist neben den Refer- | | | |
| { | enzen und Attributen die Möglichkeit, beim Unit Test auch Parameter zu übergeben | | | |
| public static string ConvertStrings | Projektreferenz: Microsoft.Pex.Framework.dll einbinden | | | |
| (string one, string two) { | in das Testprojekt einbinden. Namespace: using Microsoft.Pex.Framework; | | | |
| <pre>string result = one[2].ToString();</pre> | Attribut Testklasse: [PexClass (typeof(TestClass))] Attribute Testmethode: [PexMethod] | | | |
| result += two.substring(0, 3); return result; | Sample | | | |
| } | using System; | | | |
| } | using System.Collections.Generic; | | | |
| Überblick über die unterschiedlichen Unit Test Framework um diesen | using System.Linq; using System.Text: | | | |
| Code zu testen: | using Microsoft.Pex.Framework; | | | |
| Microsoft Visual Studio | namespace PexSample.Pex.Test | | | |
| Projektreferenz: Microsoft.VisualStudio.QualityTools.UnitTestFramework | [PexClass(typeof(Tools))] | | | |
| Attribut Testklasse: [TestClass] | public partial class ToolsTest | | | |
| Attribute Testklasse Konstruktor: [ClassInitialize()],[TestInitialize()] Attribute Testklasse DeKonstruktor: [ClassCleanup()],[TestCleanup()] | { [PexMethod] | | | |
| Attribute Testmethode: [TestMethod] | <pre>public void _1_Test(string one, string two)</pre> | | | |
| sample | { | | | |
| using System; using System Collections Generic: | (one, two); | | | |
| using System.Linq; | } | | | |
| using System.Text; | } | | | |
| using Microsoft.VisualStudio.TestTools.UnitTesting; | Im Contextmenü der Testmethode findet man <i>Run Pex Explorations</i> , um da- mit die Tests zu starten. | | | |
| | namespace PexSample.Pex.Test | | | |
| [TestClass] | { [PexClass(typeof:Tools))] | | | |
| public class lestvisualStudioPexSample { | public partial class ToolsTest { | | | |
| [TestMethod] | <pre>public mid 1 Teer (string one, string two)</pre> | | | |
| <pre>public void _1_Test() {</pre> | Pex . Tools.ConvertStrings(one, two); | | | |
| <pre>string result = PexSample.Tools.ConvertStrings</pre> |) Organze Usings | | | |
| ("Hello", "world"); | Grafik von | | | |
| <pre>Assert.IsNotNull(result, "Result is empty."); }</pre> | http://research.microsoft.com/en-us/projects/Pex/ | | | |
| } | Die ConvertStrings Methode wird analysiert, die Tests werden nun auto- | | | |
| } | matisch erstellt. Die erstellten Tests setzen auf denen von | | | |
| | visuaistudio.lestiools aur. U.n. in das lestprojekt muss auch die Keferenz | | | |
| | | | | |
| Thomas Reinwart | office@reinwart.comPENEW5-114Juni2009 | | | |
| | | | | |



| <pre>pends p</pre> | auf Microsoft.VisualStudio.QualityTools.UnitTestFramework eingebunden | PEX hat einen Fehler gefunden: | | | |
|--|---|--|--|--|--|
| <pre>PrexSomple.Pex.lest Properties References ToolsTest.cs ToolsTest.cs Doblect.lest.cs ToolsTest.cs Tools</pre> | sein. | Per Exploration Reads - stopped - I non I exploration: Durinestop for Noreskepet (Dring) 2 b + W - III () - by Vers - | | | |
| <pre>Properties References ToolsTest.c</pre> | 🖃 🥵 PexSample.Pex.Test | Review bood issues: UnMonitoria. Calobal twents: All tests: All tests: a litetatability: (%) Expect the state of the second st | | | |
| <pre>Winter State Control State Stat</pre> | 😟 🔝 Properties | G Schooles And Control Resources only and System Bolean System JD File Exist(System Sping path) | | | |
| <pre>Value of the index of the</pre> | | This method cause steatishiby problems, and as a result Pec may not be able to generate inputs that cover the code following the call. See the help for further information. | | | |
| <pre>Industry in the interval of the second second</pre> | | (1) Stack trace | | | |
| <pre>Construct with the content of this file annually.</pre> | | A giver duradourie text inectou Sets 10 * @ rep | | | |
| Es wurde ein Fehlerfall entdeckt: Teremented aus Todaries automated trade of Upper Automated Teremented Todaries automated trade of Upper Automated Teremented Teremented Todaries automated trade of Upper Automated Teremented Terem | ToolsTest1_Test.g.cs | 1 exploration: DusinessLop: ProcessRequest(String) | | | |
| <pre>line control is in the contains atomatically generated unit tests. // this file contains atomatically generated unit tests. // file contains atomatically generated unit tests. // file contains atomatically generated unit tests. // company=MyCompany*Comp</pre> | Es wurde ein Fehlerfall entdeckt: | target regfile Summary Exception Error Message 1 new Businerd.ogic() ** TypeInitializationException The type initializer three an exception. | | | |
| <pre>Im Output sele ich, das der bloße Aufruf mit einem leren Parameter i</pre> | Pex Exploration Results - stopped - 1 failed, 1 run 1 exploration: TookTest 1 Test/String one. String hum) | 2 2 new BusinessLogs() "10" TypeInitializationException The type initializer three an exception. | | | |
| <pre>wei</pre> | Review bold issues: All Tests All Events 1 Uninstrumented Method | Im Output sehe ich, das der bloße Aufruf mit einem leeren Parameter | | | |
| <pre>Pex hat erkannt, dass die Parameter über mult, mult einen Fehler in ConvertStrings liefert. Die erstellte Klasse: // <copyright file="Toolsest.lest.ge:s<br">company="MyCompany">Copyright> // <copyright file="Toolsest.lest.ge:s<br">company=MyCompany">Copyright> // <copyright file="Toolsest.lest.ge:s<br">company=MyCompany">Copyright> // <copyright file="Toolsest.lest.ge:s<br">company=MyCompany=Copyright> // <copyright file="Toolsest.lest.lest.com<br">// compleamented> // <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <copyright file="Toolsest.lest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <copyright file="Toolsest.lest.com<br">// <c< th=""><th>one two Summary/Exception Error Message</th><th>(nu11) zu einem Fehler führt, nämlich bei der Uberprüfung FileExists. Um</th></c<></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></copyright></pre> | one two Summary/Exception Error Message | (nu11) zu einem Fehler führt, nämlich bei der Uberprüfung FileExists. Um | | | |
| <pre>rekaine examine, dass die Praintieten uben mut, mut einen Penier in convertstrings liefert. Die erstellte Klasse: // coopyright file="Toolstest.l_Test.g.cs"</pre> | Dev hat orkannt dass die Darameter über will will eine objektinkalt leitigerige. | Unit Test für den Fehler schreiben, dann beheben) aufzunehmen, kann | | | |
| <pre>Steller, dass er kein weiteres Mal auftritt. Oder ich erkenne den Fehler nicht sofort und muss ohnehin debuggen, was ich erst im generierten // cauto-generated* // This file contains automatically generated unit tests. // I the indit wits file anaulty. // When Pex is invoked again. // If the contents of this file becomes outdated, e.g. if it does not // compile anymore, you may delete this file and invoke Pex again. // <fuel again.<br="" and="" anymore,="" compile="" delete="" file="" invoke="" may="" pex="" this="" you="">// <fuel compensated*<br="">using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { [TestMethod] [PexRiasedException(typeof(NullReferenceException))] public partial class ToolsTest { ftisl_Test((string)null, (string)null); } } // S2 Sammle? // Samml</fuel></fuel></pre> | ConvertStrings liefert. | ich mir den Test generieren lassen, den Fehler beheben und somit sicher | | | |
| <pre>// coupyright file="foolstest_l_test.cos" company="%yCompany">Copyright > by ThisProject 2009 // cauto-generated // This file contains automatically generated unit tests. // Do NOT modify this file manually. // When Pex is invoked again, // it might renove or update any previously generated unit tests. // If the contents of this file becomes outdated, e.g. if it does not // compile anymore, you may delete this file and invoke Pex again. // </pre> // Summary>This class contains parameterized Unit Test Studes Der von PEX erstellte Code: /// Summary>This class contains parameterized unit tests for BusinessLogic Summary>This class contains parameterized unit test studes | Die erstellte Klasse: | stellen, dass er kein weiteres Mal auftritt. Oder ich erkenne den Fehler | | | |
| <pre>Test Kain (_Credie pointine bit does) This file contains automatically generated unit tests. This file contains for management tests for ma</pre> | // <copyright <="" file="ToolsTest. 1 Test.g.cs" th=""><th>nicht sofort und muss ohnehin debuggen, was ich erst im generierten</th></copyright> | nicht sofort und muss ohnehin debuggen, was ich erst im generierten | | | |
| <pre>// 'Auto-generated> // This file contains automatically generated unit tests. // Do NOT modify this file annually. // When Pex is invoked again, // it might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update any previously generated unit tests. // // It might remove or update this file and invoke Pex again. // </pre> /// <summary>This class contains parameterized unit tests for BusinessLogic/Summary> Der von PEX erstellte Code: /// <summary>Tis class for BusinessLogic()] [PextlisedExceptionFromTypeUnderTest (typeof(AugumentException), AcceptExceptionFromTypeUnderTest (typeof(InvalidoperationException))] public partial class ToolsTest { [FexAlisedException(typeof(NullReferenceException))] public void _l_Test((string)null, (string)null); } 22 Sammle?</summary></summary> | <pre>company="MyCompany">Copyright © by ThisProject 2009 //</pre> | | | | |
| <pre>// Do NOT modify this file manually. // // When Pex is invoked again, // it might remove or update any previously generated unit tests. // // If the contents of this file becomes outdated, e.g. if it does not // compile anymore, you may delete this file and invoke Pex again. // </pre> // Summary>This class contains parameterized unit tests for BusinessLogic [restClass] using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { [TestMethod] [PexAGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_TestOl() { this1_Test((string)null, (string)null); } } // 22 Sammle? } Pex Pex Create Parameterized Unit Test Stubs Der von PEX erstellte Code: /// <summary>This class contains parameterized unit tests for BusinessLogic</summary> [restClass] [PexClass(typeof(BusinessLogic))] [PexAllowedExceptionFromTypeUnderTest (typeof(InvalidOperationException))] public void _1_TestOl() { this1_Test((string)null, (string)null); } } // Summary2 // Code: /// Code: // Code: /// summary2 [PexAmethod] ([PexAssumeUnderTest]BusinessLogic Test.ProcessRequest // (BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } } | <pre>// <auto-generated> // This file contains automatically generated unit tests.</auto-generated></pre> | Run Pex Explorations | | | |
| <pre>/// When Pex is invoked again, // It might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ it might remove or update any previously generated unit tests. // (/ is class contains parameterized unit tests for BusinessLogic</pre> | <pre>// Do NOT modify this file manually.</pre> | Pex Create Parameterized Unit Test Stubs | | | |
| <pre>// it might remove or update any previously generated unit tests. // // it might remove or update any previously generated unit tests. // // compile anymore, you may delete this file and invoke Pex again. // using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { [restMethod] [PexCaisedException(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sammle? Z2 Sammle? Der von PEX erstellte Code: /// <summary>This class contains parameterized unit tests for BusinessLogic /// summary>This class contains parameterized unit tests for BusinessLogic /// summary>Test ([restMethod] [PexRaisedException(typeof(ToolsTest))] [public void _1_Test01() { (this1_Test((string)null, (string)null); } } // 28 Sammle? // 28 Sam</summary></pre> | // // When Pex is invoked again, | | | | |
| <pre>/// If the contents of this file becomes outdated, e.g. if it does not // compile anymore, you may delete this file and invoke Pex again. // using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { public partial class ToolsTest { [restMethod] [PexRaisedException(typeof(NullReferenceException))] public void _1_TestOl() { this1_Test((string)null, (string)null); } } 22 Sammle2 22 Sammle2 /// If the contents of this file becomes outdated, e.g. if it does not /// <summary>This class contains parameterized unit tests for BusinessLogic</summary> [restClass] [PexClass(typeof(BusinessLogic))] [PexRaisedException[summary>Test studion=Test (typeof(ArgumentException))] public void _1_TestOl() { this1Test((string)null, (string)null); } } } // 22 Sammle2 } } </pre> | <pre>// it might remove or update any previously generated unit tests. //</pre> | Der von PEX erstellte Code: | | | |
| <pre>// compile anymore, you may delete this file and invoke Pex again. // using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { public partial class ToolsTest { [TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_TestOl() { this1_Test((string)null, (string)null); } } // Z2 Sammle2</pre> | $^{\prime\prime}$ // If the contents of this file becomes outdated, e.g. if it does not | /// <summary>This class contains parameterized unit tests for</summary> | | | |
| <pre>using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { public partial class ToolsTest { [TestMethod] [PexGeneratedBy (typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sammle2</pre> <pre> PexClass(typeof(BusinessLogic))] [PexClass(typeof(BusinessLogic))] [P</pre> | <pre>// compile anymore, you may delete this file and invoke Pex again. // </pre> | [TestClass] | | | |
| <pre>using Microsoft.VisualStudio.TestTools.UnitTesting; using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { public partial class ToolsTest { [TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } } 22 Samule2 </pre> [PexAllowedExceptionFromTypeUnderTest (typeof(ArgumentException), AcceptExceptionSubtypes = true)] [PexAllowedExceptionFromTypeUnderTest (typeof(InvalidOperationException))] public partial class BusinessLogicTest { /// <summary>Test stub for ProcessRequest (String)</summary> [PexMethod] public bool ProcessRequest ([PexAssumeUnderTest]BusinessLogic target, string reqFile) { // TOD0: add assertions to method // BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } } | using System; | <pre>[PexClass(typeof(BusinessLogic))]</pre> | | | |
| <pre>using Microsoft.Pex.Framework.Generated; namespace PexSample.Pex.Test { public partial class ToolsTest { [TestMethod] [PexAaisedException(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } } 22 Sample2 </pre> | <pre>using Microsoft.VisualStudio.TestTools.UnitTesting;</pre> | [PexAllowedExceptionFromTypeUnderTest (typeof(ArgumentException), | | | |
| <pre>namespace PexSample.Pex.Test { public partial class ToolsTest { [TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } } 22 Sample2 } } (typeof(InvalidOperationException))] </pre> | using Microsoft.Pex.Framework.Generated; | [PexAllowedExceptionFromTypeUnderTest | | | |
| <pre>public partial class ToolsTest { public partial class ToolsTest { [TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } // TOD0: add assertions to method // BusinessLogicTest.ProcessRequest(reqFile); return result; } 22 Sample2 </pre> | namespace PexSample.Pex.Test | <pre>(typeof(InvalidOperationException))]</pre> | | | |
| <pre>{ { { [TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sample? } } }</pre> | public partial class ToolsTest | public partial class BusinessLogicTest | | | |
| <pre>[TestMethod] [PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sample2</pre> [PexMethod] public bool ProcessRequest ([PexAssumeUnderTest]BusinessLogic target, string reqFile) { // TOD0: add assertions to method // BusinessLogicTest.ProcessRequest // (BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } | { | <pre>1 /// <summarv>Test stub for ProcessReguest(String)</summarv></pre> | | | |
| <pre>[PexGeneratedBy(typeof(ToolsTest))] [PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sample? </pre> public bool ProcessRequest ([PexAssumeUnderTest]BusinessLogic target, string reqFile) { // TOD0: add assertions to method // BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } | [TestMethod] | [PexMethod] | | | |
| <pre>[PexRaisedException(typeof(NullReferenceException))] public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sample? </pre> ([PexRaimeUnderTest]BusinessLogic target, string redFile) { // TOD0: add assertions to method // BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } | <pre>[PexGeneratedBy(typeof(ToolsTest))]</pre> | public bool ProcessRequest | | | |
| <pre>public void _1_Test01() { this1_Test((string)null, (string)null); } } 22 Sample? </pre> // TOD0: add assertions to method // BusinessLogicTest.ProcessRequest // (BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } | [PexRaisedException(typeof(NullReferenceException))] | <pre>([PexAssumeUnderTest]BusinessLogic target, String reqFile) {</pre> | | | |
| <pre>this1_Test((string)null, (string)null); } // BusinessLogicTest.ProcessRequest // (BusinessLogic, String) bool result = target.ProcessRequest(reqFile); return result; } 22 Sample? }</pre> | <pre>public void _1_Test01() {</pre> | <pre>// TODO: add assertions to method</pre> | | | |
| <pre>} // (businessLogic, string) bool result = target.ProcessRequest(reqFile); return result; } 22 Sample? }</pre> | this. 1 Test((string)null, (string)null); | <pre>// BusinessLogicTest.ProcessRequest // (RusinessLegic_String)</pre> | | | |
| <pre>} return result; } 22 Sample 2 }</pre> | } | bool result = target.ProcessReguest(regFile): | | | |
| 22 Samule 2 | 3 | return result; | | | |
| | 22 Sample 2 | } | | | |

Wietestet PEX? Was bringt mir das in der Praxis - Beispiel des Einsatzes der PEX Explora-

| | | oration Results - | streped - | 2 falled, 3 runs | | | |
|-----|------|-------------------|------------|---------------------|---|-----------------------------|---------------------|
| 1 e | plor | ation: Busines | sLogic Pro | cessRequest(String) | - In | Views * | |
| Re | vice | bold issues: | All Tests | 2 Failed Tests | All Events 🔒 1 Testability 🕚 3 Uninst | rumented Methods 🐴 I Object | Creation |
| | | target | | reqfile | Summary/Exception | Error Message | |
| 4 | 1 | new Busines | sLogic() | | TypeInitializationException | The type initializer | threw an exception. |
| 4 | 2 | new Busines | sLoge() | -/0- | TypeInitializationException | The type initializer | threw an exception. |

Um nicht alle möglichen Zeichen (65535) durchzutesten (das würde zu lange dauern), testet PEX jeweils ein Zeichen eines bestimmten Typs. (Also Klein, Gross usw.) Wenn hier Fehler auftreten, wird der Wert der den Fehler verursacht, mit der Art der Exception aufgelistet. In den Details ist dann die Stelle im Code zu erkennen.

3. Links

http://research.microsoft.com/en-us/projects/Pex/

http://research.microsoft.com/en-us/people/jhalleux/

4. Fazit

Mit dieser Technik wird das Testen wesentlich effektiver und die Komponenten sind viel besser getestet. Der Vorteil zu den selbsterstellten Unit Tests ist, dass Pex eine Codeanalyse des zu testenden Codes vornimmt und wirksame Tests generiert, also beispielsweise mit den Randbereichen der Übergabeparameter. Somit ist der erstellte Code automatisch wesentlich besser und schneller getestet, als dies bisher manuell möglich ist.

25

if (flyrtens 10.File.Exists(regFile)) Ligger Log. Debug (string.Format ("De return false; es: 🥥 could not find Cleanup Code Refator Run Unit Testa Č.

Bei dieser Methode einer Klasse wird eine Datei als Parameter übergeben. Es gibt die Abfrage File.Exists im Code, auf den ersten Blick sieht der Code sauber aus. Die bestehenden Unit Test, die Mock Files als Parameter verwenden, sind alle fehlerfrei. Was erkennt PEX, gibt es doch

Auf der Methode selber starte ich im Contextmenü "Run Pex Explorations"

tion im bestehenden Code:

noch Fehler im Code?

pic.cs*

und warte gespannt einige Sekunden.

cpaces name="reqFile")-d/parent
lin bool ProcessRequest(string reqFile)

System.Diagnostics.Stonwatch stopWatch stopWatch.Start();