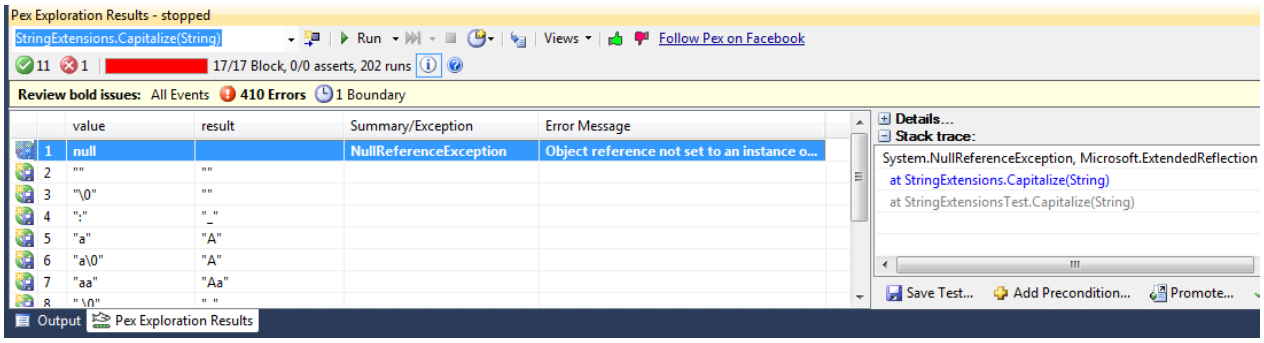
5. gyakorlat: Automatikus tesztgenerálás

A gyakorlat során a **Microsoft Pex** eszközt fogjuk megismerni. A Pex segítségével automatikusan lehet tesztbemeneteket generálni .NET kódhoz. A Pexről volt szó előadáson, ez is egy szimbolikus végrehajtást használó eszköz, ami megpróbál olyan bemeneteket kiválasztani, amik a tesztelt kód minél nagyobb részét fedik le.

A Pex úgynevezett *paraméteres egységteszt*eket (Parameterized Unit Test – PUT) használ: ezek egyszerűen olyan teszt metódusok, amiknek paramétere van, így különböző tesztbemenetek használatával több, hasonló jellegű ellenőrzést tudunk velük elvégezni[1]. A PUT-ban végrehajtjuk a szokásos Act/Arrange/Assert lépéseket, azonban a teszt paramétereit nem kell nekünk kézzel meghatározni és megadni, azt majd a Pex kiválasztja egy úgynevezett *felderítés* (exploration) során.

A Pex használható parancssorból vagy Visual Studio bővítményként is (1. ábra). A jelenlegi stabil verzió csak Visual Studio 2010-zel működik együtt, a gyakorlaton ezt fogjuk használni. (A nemrég megjelent Visual Studio 2015 betába már bekerült a Pex.)



1. ábra: Pex Exploration Results ablak

A Pex alapvető képességeinek használata

Az első részben a Pex alapfunkcióival fogunk megismerkedni. Ehhez a Pex hivatalos útmutatóját fogjuk használni, ami megtalálható a Pex könyvtárában a virtuális gépen (C:\Program Files\Microsoft Pex\Documentation).

A következő leírásra lesz szükség:

Exploring Code with Microsoft Pex.docx

Az útmutató 4 egyszerű gyakorlaton keresztül vezet végig. Végezzük el egyesével ezeket!

1. *Exercise 1: Running Pex for the First Time*
   - Elsőként egy egyszerű `Capitalize` metódust kell implementálni, ami egy string átalakítását végzi el. A metódusnak nincs semmi függősége, így könnyű tesztelni.
   - A létrehozott kódon futtatni kell a Pex felderítését. A Pex bejárja a kódot, és olyan bemeneteket választ ki, amivel az összes utasítást és ágat le tudja fedni.

[1] A korábbi gyakorlatok során a JUnit-nál is találkoztunk már hasonló fogalommal.

  - (A felderítés során „runtime context … failed to open” hibákat fog dobálni, ezt meg lehet kerülni így[2].)
  - A felderítés során nem kezelt kivételt is talál, majd erre ajánlást is tesz, hogy hogyan lehetne elkerülni.

- *Exercise 2: Saving a Test, and Debugging Issues*
  - A korábbi felfedezések eredményeit fogjuk elmenteni egységtesztekbe. Létrejön egy paraméteres teszt, majd a Pex elmenti külön-külön az összes kiválasztott bemeneti értéket olyan sima tesztesetekként, amik a PUT-ot hívják.
- *Exercise 3: Confirming Code Coverage with Team Test*
  - Mivel a Visual Studio Premium verziója van fent, ezért tudjuk használni annak a kódfedés mérést végző komponensét.
  - Ha kitöröljük például a null-os esetet a generált tesztkódból, akkor egy újrafuttatás után látjuk is, hogy mér és tudja a forráskódot is színezni.
- *Exercise 4: Taking a First Look at Parameterized Unit Tests*
  - Utolsó lépésként megnézzük a PUT kódját, és kibővítjük további Assert utasításokkal, hogy a tesztjeink a funkcionalitást is, és ne csak a nem kezelt kivételeket tudják ellenőrizni.

Ez az egyszerű példa bemutatta a Pex alapvető képességeit, de itt még semmi korlátba nem ütköztünk.

A Pex működése

Nézzünk bele kicsit a Pex belső működésébe! A Pex meg tud jeleníteni mindenféle részletet a felderítésről:

- A beállítások között (*Tools / Options / Pex / General*) kapcsoljuk be a diagnosztikai információt (Diagnostic) és a jelentéskészítést (Report).
- A teszt projektből töröltessük ki a korábban generált teszteket, majd futtassunk egy felderítést az új beállításokkal.
- A projekt *bin* könyvtárában lévő *reports* alkönyvtárból keressük ki az utolsót, és nyissuk meg a *report.html* fájlt.
- Kattintsunk a teszt nevére (StringExtensionsTests), és itt nézzük meg a linkek segítségével a generált paraméterértékeket, a grafikonokat a kódfedés növekedéséről és magát a dinamikus kódfedést is.
- Nézzük meg most a *log* nevű linket, itt látszanak az egyes megtalált tesztesetek kódjai. Kattintsunk itt a *Path condition* melletti + jelre, ilyenkor látszik az adott lefutáshoz tartozó feltételrendszer. Próbáljunk egy ilyet végigkövetni a kódban!

A Pex működésének megértésében a Pex egy másik leírása fog segíteni:

[2] http://stackoverflow.com/questions/17486001/error-execution-runtime-context-during-pex-exploration

Advanced Concepts: Parameterized Unit Testing with Microsoft Pex [3]

Ez a következő részből áll:

- Az „Understanding Whitebox Software Testing” fejezetben szereplő fogalmak már ismerősek az előadásról, ezt átpörgethetjük, a Program 1-et nézzük meg belőle.
- Az „Understanding Pex” feladatait érdemes végigcsinálni:
  - A példaprogramokat belerakhatjuk az első részben elkészült projektbe. Érdemes a metódusokat statikussá tenni, és akkor nem kell bajlódni azzal, hogy hogyan példányosítsa a Pex.
  - A kódokon futtassuk le a Pex felfedezését, és ne csak megnézzük az eredményt a leírásban!
  - Exercise 1: a Program 1-et vizsgáljuk, a Pex megfelelő hívásaival a kódból is meg tudjuk nézni a *path constraint* értékét, valamint a szimbolikus változókat.
  - Exercise 2: egy egyszerű objektumon keresztül vizsgáljuk, hogy hogyan tárolja el a Pex belül a struktúrákat és objektumokat.
  - Exercise 3: a felfedezés során a lehetséges különböző utakat más-más stratégia szerint járjuk be.
  - Exercise 4: egy egyszerű példán keresztül megnézzük, hogy milyen kényszerhalmazokat tud a Pex megoldatni.
  - Exercise 5: példák, hogy hogyan próbál a Pex nem kezelt kivételeket találni a kódban (úgynevezett implicit ágak segítségével).

Ezek a feladatok alapján remélhetőleg kaptunk egy magas szintű képet, hogy hogyan is működnek a dinamikus szimbolikus végrehajtást használó tesztbemenet-generáló eszközök.

További információ

A Pex telepítőkészletében vagy a weboldalon további leírások is találhatóak, pl.

- *Parameterized Unit Testing with Microsoft Pex*: az eleje egy jó összefoglaló az egységtesztelésről és a TDD-ről, példákon keresztül bemutatja, hogy hogyan működik ez a Visual Studio esetén. Utána a Pex és a Moles használatára mutat egy-egy példát.
- *Honfi Dávid, Szimbolikus végrehajtás alapú tesztgenerálás támogatása és analízise, BME VIK, TDK dolgozat, 2014:* a szimbolikus végrehajtás működésének megértését és a felderítés során kapott problémák megoldását segítő módszer és eszköz fejlesztése.

[3] Elérhető: http://research.microsoft.com/en-us/projects/pex/pexconcepts.docx