Pester 101

Rinon Belegu

Agenda

- Was ist Pester?
- Test Driven Development (TDD)
  - Vorteile
  - Warum ist Testen wichtig?
- Grundlagen Pester
  - Describe Block
  - Assertion
  - Mock
  - Demo

Was ist Pester?

- Test Framework für PowerShell Unit Test
- Ermöglicht das automatische testen des Codes
- Ermöglicht Test Driven Development (TDD)
- Eigentlich «nur» ein PowerShell Script

Unit Testing

- White-Box Test «Code ist bekannt»
- Alle Funktionen Minimum ein mal ausgeführt
- Die Soll-Ergebnisse werden getestet
- Möglichst wenig Testfälle angepeilt

Unit Test - Ablauf

- Initialisierung
- Vergleich Ist-Wert mit Soll-Wert
- Aufräumen (CleanUp)

Test Driven Development

- Traditionell: Zuerst Code dan Test
- TDD -> kehrt Prozess um
- Die funktionale Spezifikation des Codes als Test
- Nachher wird der Code geschrieben

Vorteile TDD

- Denken vor Scripten
- Problem erkannt?
- Kohäsion / Kopplung
- Hilft bei Clean Code
- Qualität des Codes / Produkts ist höher

Kohäsion / Kopplung

- Kohäsion
  - Methode, Klasse oder Modul ist genau für eine Aufgabe
- Kopplung
  - Klassen / Module sollten so wenig «voneinander» Wissen wie möglich

Warum ist Testen wichtig?

- Dokumentation
- Komplexität der Lösungen wird immer höher
- Zuverlässigkeit des Codes

Describe-Block

- Beschreibt was getestet werden soll
- Grösster Container für Tests
- Unterteilbar in kleinere Test-Blöcke «Context»
- Der Describe-Block und Context Block sind «Bereiche»
- It-Blöcke stellen einen Task/Bedingung dar

Describe-Block – Syntax

```
Describe 'beschreibung' [-Tags 'Unit'] {
    Context 'teil 1' {
        it 'macht was' {
        Get-Something |Should be $true
    }
  }
}
```

Describe-Block - Tag

```
Describe -Tag 'TeilTest' "Mach-Was" {}

Invoke-Pester -Tag 'TeilTest','TestZwei'
Invoke-Pester -ExcludeTag 'TeilTest'
z.B. Tag  "RequireAdminOnWindows"
```

Assertion - Behauptung

- Stellt Behauptung auf
- Methode in Pester um Vergleiche zu ziehen

Assertion - Umgebungsvorbereitung

- BeforeAll, Afterall -> Vor oder nach allen It Block
- BeforeEach,AfterEach -> Vor und nach jedem It Block

Assertion - Umgebungsvorbereitung

```
Describe "Demo" {
    BeforeEach {
        "Erster"
    }
    Context "Untercontext" {
        BeforeEach { "zweiter" }
        It "Macht ne demo" {
            "dritter" | Should BeOfType System.String
    }
  }
}
```

https://github.com/pester/Pester/wiki/BeforeEach-and-AfterEach

TestDrive

- Temporäres Filesystem (Temp Ordner)
- Zugreifbar über TESTDRIVE:
- Wird nach jedem Test gelöscht

Pester nimmt an das..

- Code in einer .ps1 File
- Code als Function (Script auch möglich)
- Funktions Ausgabe über Write-Output
- Deterministischer Code -> geicher Input / gleicher Output
- Input über Parameter

Mögliche Testergebnisse

- Passed
- Failed
- Skipped
- Pending (leer oder markiert mit -Pending)

Mocks

- Eigener Code abhängig von aufruf Fremdcode
- Anstatt den Fremdcode zu testen z.B. Get-Item
- Liefert vordefinierte Ausgabe

Mock - Syntax

```
Mock Get-Item {
   'Item'
}
```

Fremde Funktionen «aufrufen» nur Rückgabewert «simulieren»

https://github.com/pester/Pester/wiki/Mock

Demo

Summary

- Denken vor Skripten
- Funktionale Anforderungen sauber definiert
- Hilft Code zu reduzieren

Links

- https://de.wikipedia.org/wiki/Modultest

Questions?

About_Author

- Get-Help About_Author
- Rinon Belegu
- Senior System Engineer - UMB AG

- @BelegуRinon (Twitter)
- Mail: rinon@belegu.ch

- Watch out:
  - www.get-powershell.ch

Next Steps...

- Now: 15 min break

- Grab a coffee
- Stay here to enjoy next presentation
- Change track and switch to another room

- Ask me questions or meet me in a breakout session room afterwards

psconf.eu 2018

scheduled to be in the week of

April 16-20, 2018

details on www.psconf.eu as they become available