Programowanie Defensywne

Defensive Programing

… an approach to improve software and source code, in terms of:

- General quality - reducing the number of software bugs and problems.
- Making the source code comprehensible - the source code should be readable and understandable so it is approved in a code audit.
- Making the software behave in a predictable manner despite unexpected inputs or user actions.

Wikipedia 30.09.2015r.

Programowanie Defensywne

… jest to podejście, które ma na celu poprawę działania oprogramowania i jego kodu źródłowego pod względem:

- Jakości – redukcja ilości błędów oraz problemów.
- Zrozumienia kodu źródłowego – kod źródłowy powinien być czytelny.
- Tworzenie oprogramowania w taki sposób, by zachowywało się ono zawsze w sposób przewidywalny, pomimo nieznanych danych wejściowych oraz działań użytkownika.

Wikipedia 30.09.2015r.

Expect the Unexpected

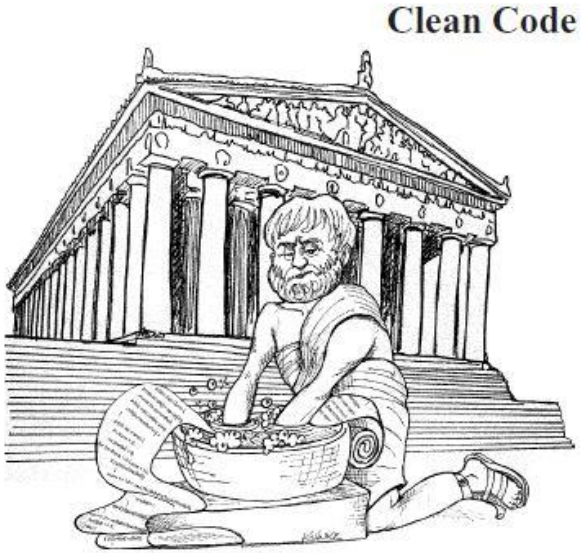
The whole point of defensive programming is guarding against errors you don’t expect.

Steve McConnell, Code Complete

ilustracja, źródło: http://www.mariowiki.com/File:MnL_BeanbeanGuard.png

Jakość oraz Czytelność

- „Czysty kod” (ang. Clean Code)
  = łatwy do zrozumienia i łatwy do zmiany
- Testowalny kod
- Reguły budowania oprogramowania,
  które zachowuje się
  w przewidywalny sposób



Ilustracja z książki Clean Code, Robert Cecil Martin

Podstawowe zasady programowania defensywnego (*Steve McConnell*)

- **1. Walidacja danych pochodzących „zewnątrz”**
- **2. Obsługa błędnych danych**
- **3. Nieufność dla obcego kodu**
- **4. Dokumentacja założeń (assumptions)**
- **5. Standaryzacja i strategia obsługi wyjątków**
- **6. Bezpośrednia obsługa wyjątku, tylko jeśli jest konieczna**
- **7. Diagnostyka**

1. Walidacja danych pochodzących z „zewnątrz”

- **Określamy granice zaufania**
  - » Dane wprowadzone przez użytkownika, pliki, siec, itd..
- **Weryfikacja parametrów wejściowych**
  - » Poprawność typów, zakres danych, null, czy stan obiektu jest spójny
- **Czujność dla nietypowych sytuacji**

2. Jak sobie radzić ze złymi danymi

Metoda()

ŹLE !!

Ilustracja: http://www.cliparthut.com/trash-can-clip-art-clipart-WaGqIc.html

2. Jak sobie radzić ze złymi danymi

Metoda()

Nothing out

Exception out

Return Codes

Status Parameter

Error Message

Przykład

```
public decimal CalculateProcentOfGoalStepsBad(string goalStep, string actualSteps)
{
    return decimal.Parse(goalStep) / decimal.Parse(actualSteps) * 100;
}
```

```
public decimal CalculateProcentOfGoalStepsGood(string goalStep, string actualSteps)
{
    decimal goalStepCount,actualStepsCount;

    if (string.IsNullOrWhiteSpace(goalStep))
        throw new ArgumentException("Goal step must be entered","goalStep");

    if (string.IsNullOrWhiteSpace(actualSteps))
        throw new ArgumentException("Actual steps must be entered", "actualSteps");

    if (!decimal.TryParse(goalStep, out goalStepCount))
        throw new ArgumentException("Goal step mus be numeric", "goalStep");

    if (!decimal.TryParse(actualSteps, out actualStepsCount))
        throw new ArgumentException("actualSteps mus be numeric", "actualSteps");

    if (goalStepCount < 0)
        throw new ArgumentOutOfRangeException("goalStep", goalStep,"Goal must be grat

    if (goalStepCount > actualStepsCount)
        throw new ArgumentOutOfRangeException("goalStep", goalStep, "Goal must be sma

    return actualStepsCount / goalStepCount * 100;
}
```

Bezpieczeństwo

- Open Web Application Security Project:
  Top 10 Application Security Risks
  https://www.owasp.org/index.php/Top_10_2010-Main

- Common Weakness Enumeration:
  Top 25 Most Dangerous Software Errors
  http://cwe.mitre.org/top25/#CWE-20

3. „Zewnętrzny” Kod

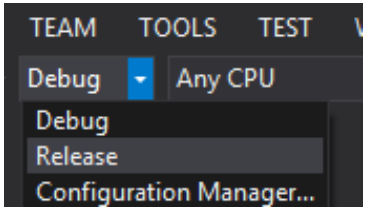
- **Nie zakładaj, że wywołanie metody spoza twojego kodu działa zgodnie z założeniami**
- **Upewnij się, że masz obsługę błędów dla zewnętrznych API oraz bibliotek**

Asercja (Assert)

- **Asercja – deklaracja w kodzie, która zawsze musi być prawdziwa**

```
public AssertExample(IRepository repository)
{
    Debug.Assert(repository!=null,"repository is
```

- **Asercje są używane tylko w trakcie tworzenia aplikacji**
  - » Usuwane z finalnego kodu (Release)



- **Asercja tylko dla kontroli warunków, które nigdy nie powinny się wydarzyć**

4. Dokumentacja Założeń (assumptions) Assert / Code Contract

```
private Item Read(int id)
{
    // precondition
    Debug.Assert(id>0);

    var item = _repository.GetItem(id);

    // postcondition
    Debug.Assert(ItemValidator.IsValid(item));
    return item;
}
```

Assert – inny przykład

```
public class AssertExample
{
    private readonly IRepository _repository;

    public AssertExample(IRepository repository)
    {
        Debug.Assert(repository!=null);

        _repository = repository;
    }

    public void Save(Item item)
    {
        _repository.SetItem(item);
    }
```

Exception vs Assert

- **Walidacja parametrów (preconditions)**
  - » Metody Publiczne -> Exception
  - » Metody Prywatne -> Assert (nie jest to reguła !)
- **Walidacja warunków Końcowych (postcondition)**
  - » Assert (C , C++)
  - » Unit Test (C# , C++)
- **Error Handling dla błędów, które oczekujemy że mogą się zdarzyć**
  Assert dla warunków, które NIGDY nie powinny wystąpić
- **Jeśli nie jesteś pewien - Exception**

5. Chwytanie wyjątków – standaryzacja i strategia

- Ustalenie standardu „chwytania” wyjątków.

  Strategia obsługi „normalnych wyjątków”, „oczekiwanych wyjątków” oraz konsekwentne stosowanie tych reguł.

- Obsługa wyjątku powinna uwzględniać:
  - Zapis danych trwałych (persistent)
  - Zamknięcie odpowiednich plików
  - Zapis do logu informacji o błędzie
  - Komunikat o błędzie dla użytkownika.

Obsługa Błędów vs Technologie (przykłady)

- Web.config / App.config
- WinForms - Application.ThreadException Event
- Asp – Global.asax / Application_Error
- ASP MVC – IExceptionFilter / RegisterGlobalFilters
- WCF
  - » FaultException, CommunicationException
  - » Service Behavior IErrorHandler
- Aspekty (AOP aspect-oriented programming)
- Enterprise Library - Exception Handling Application Block

Przykład – Enterprise Library

```
try
{
  SalaryCalculator calc = new SalaryCalculator();
}
catch (Exception e)
{
  Exception exceptionToThrow;
   if (exceptionManager.HandleException(e, "Exception Policy Name",
       out exceptionToThrow))
   {
       if(exceptionToThrow == null)
         throw;
       else
       throw exceptionToThrow;
   }
}
```

Przykład C#

```
public T ReTry<T>(Func<T> func, int reTryTimeInMiliSeconds = 100)
{
    int tryCounter = 0;
    while (true)
    {
        tryCounter++;
        try
        {
            return func();
        }
        catch (Exception)
        {
            if (tryCounter >= 3)
            {
                throw;
            }
        }
        Thread.Sleep(reTryTimeInMiliSeconds);
    }
}
```

Przykład C#

```
public DataSet Execute()
{
    return exceptionStrategy.ReTry(ExecuteSqlQuery);
}

private DataSet ExecuteSqlQuery()
{
```

6. Bezpośrednia obsługa wyjątku, tylko jeśli jest konieczna

- Jeśli nie jest to potrzebne, wyjątek nie powinien być bezpośrednio „chwytany”. Powinien zostać obsłużony przez główny mechanizm obsługi błędów.
- Obsługiwany tylko konkretny typ wyjątku

```
try
{
 calculator.CalculateProcentOfGoalSteps(inputSteps, inputActualStep);
}
catch (ArgumentException e)
{
```

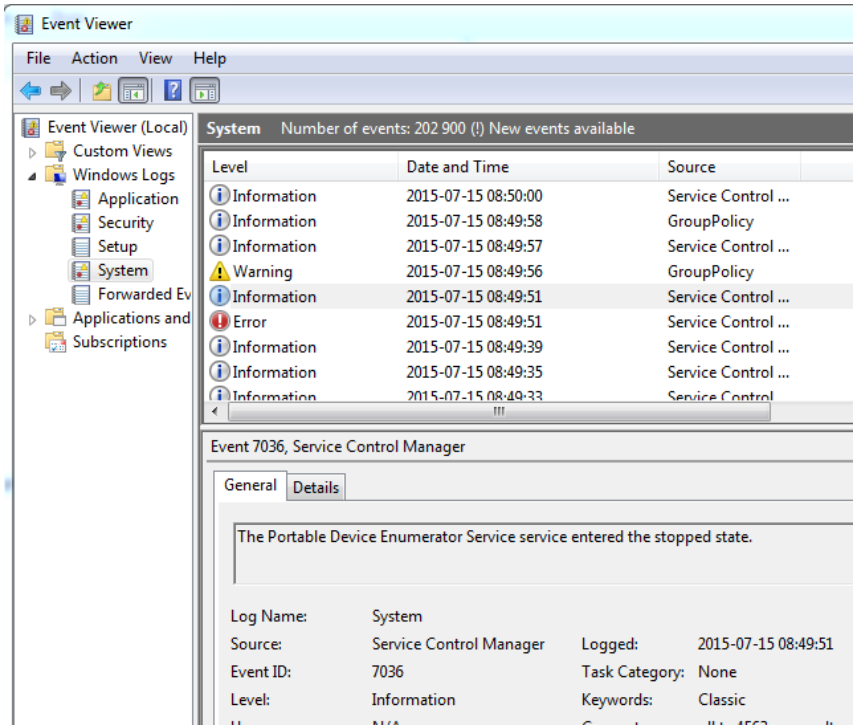
7. Diagnostyka

- Dodatkowy kod diagnostyczny
  (logging, tracing), który ułatwia instrumentalizację oraz wyjaśnienie co się dzieje z aplikacją w trakcie jej działania.

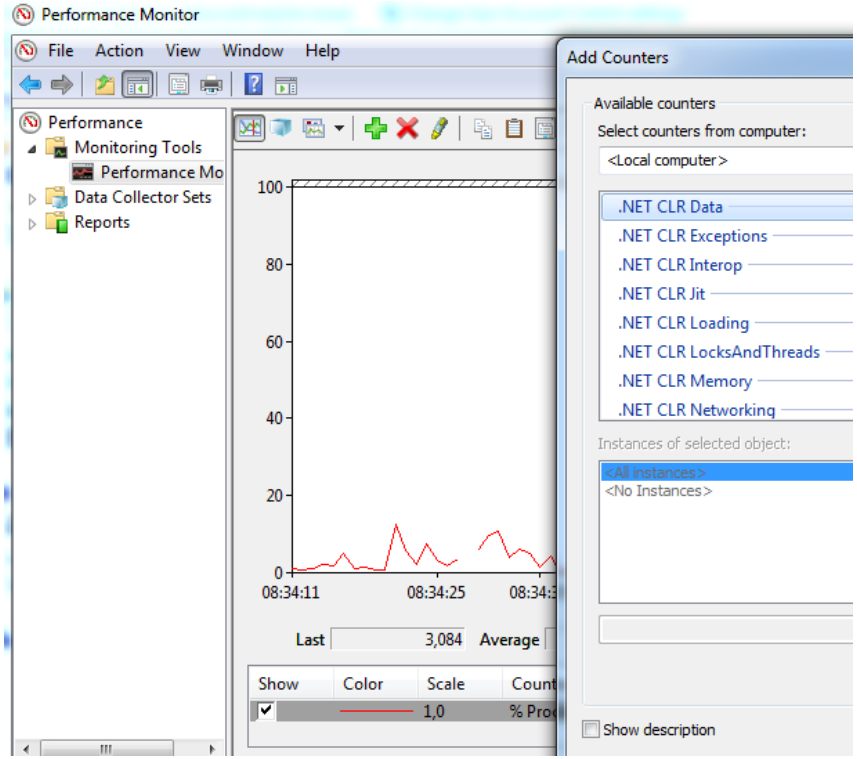
- Diagnostyka zapisywana jest:
  Output Window, pliki logów, bazy danych,
  Windows Event, Windows Performace Counters

- Uwaga!
  > Informacji Diagnostycznych
  < Wydajność aplikacji
  Ustawienie odpowiedniego poziomu diagnostyki
  (Debug/Warning/Error)

Event Viewer



Performance Monitor



Kroll Ontrack

7. Diagnostyka

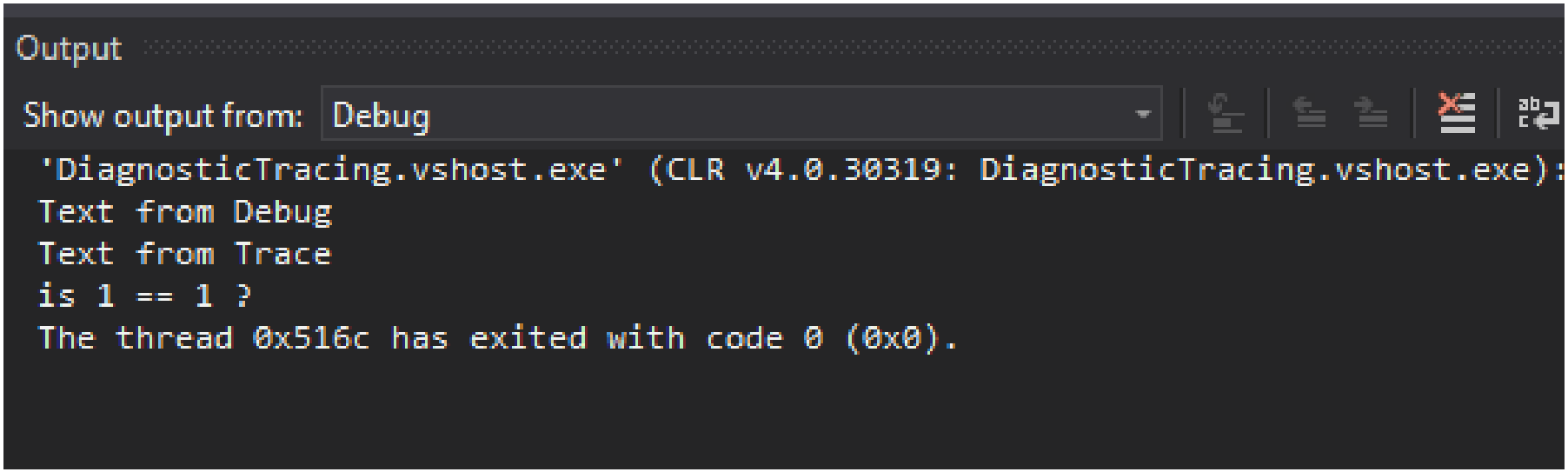
- Przykładowe narzędzia:

  .Net System.Diagnostics
  Log4net
  Nlog
  Elmah
  Serilog
  Enterprise Library - Logging Application Block

Diagnostyka – Trace / Debug Przykład

```
Debug.WriteLine("Text from Debug");
Trace.WriteLine("Text from Trace");

Trace.WriteLineIf(1==1,"is 1 == 1 ?");
```



Diagnostyka – Trace / Debug Przykład

```
<system.diagnostics>
  <trace autoflush="true">
    <listeners>
      <remove name="Default" />
      <add name="myTextTraceListner"
           type="System.Diagnostics.TextWriterTraceListener"
           initializeData="TextWriterOutput.log"
           traceOutputOptions="Callstack" >
      </add>

      <add name="myEventLogTraceListner"
           type="System.Diagnostics.EventLogTraceListener"
           initializeData="DefensiveProgramming" />
      <add name ="myConsoleTraceListner" />
    </listeners>
  </trace>
```

Diagnostyka – Log4Net Przykład

```
ILog log = LogManager.GetLogger(typeof (Log4NetExample));


log.Warn("Warning Message");

try
{
    throw new InvalidOperationException("Ups ... we have a bug");
}
catch (Exception ex)
{
    log.Error("Catched Excpetion", ex);
}
```

Diagnostyka – Log4Net Przykład

```
<log4net>
  <appender name="LogFileAppender" type="log4net.Appender.FileAppender">
    <param name="File" value="Log4net.txt" />
    <param name="AppendToFile" value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <param name="ConversionPattern" value="%d [%t] %-5p %c %m%n" />
    </layout>
  </appender>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
  </appender>
  <root>
    <level value="DEBUG" />
    <appender-ref ref="LogFileAppender" />
    <appender-ref ref="ConsoleAppender" />
  </root>
</log4net>
```

Na koniec

- **Programowanie Defensywne =**
  Bezpieczeństwo, Stabilność aplikacji, Diagnostyka

- **Wyznaczenie granic i reguł zaufania**
  Zdefiniowanie miejsc / warstw aplikacji w których się „bronimy”

- **Rozsądek !**
  Techniki programowania defensywnego powinny usprawniać,
  a nie utrudniać prace z kodem