.NET User Group Bern

Refactoring: Mythen & Fakten

Johnny Graber

http://www.JGraber.ch
JG@JGraber.ch

Follow dnugbe on twitter
http://www.dnug-bern.ch/rss.aspx

DNUG Bern Regionalsponsoren

Born Informatik

Microsoft

bbv Software Services AG

M&S

WISS
WIRTSCHAFTSINFORMATIKSCHULE SCHWEIZ

DNUG Bern Sachsponsoren

Heutiges Twitter-Hashtag

#dnugbemix

Über Johnny Graber

- SW Ingenieur @ FMH, Bern
  - E-Logbuch und Fortbildungsplattform

- Web: http://www.JGraber.ch
- Twitter: @j_graber

Agenda

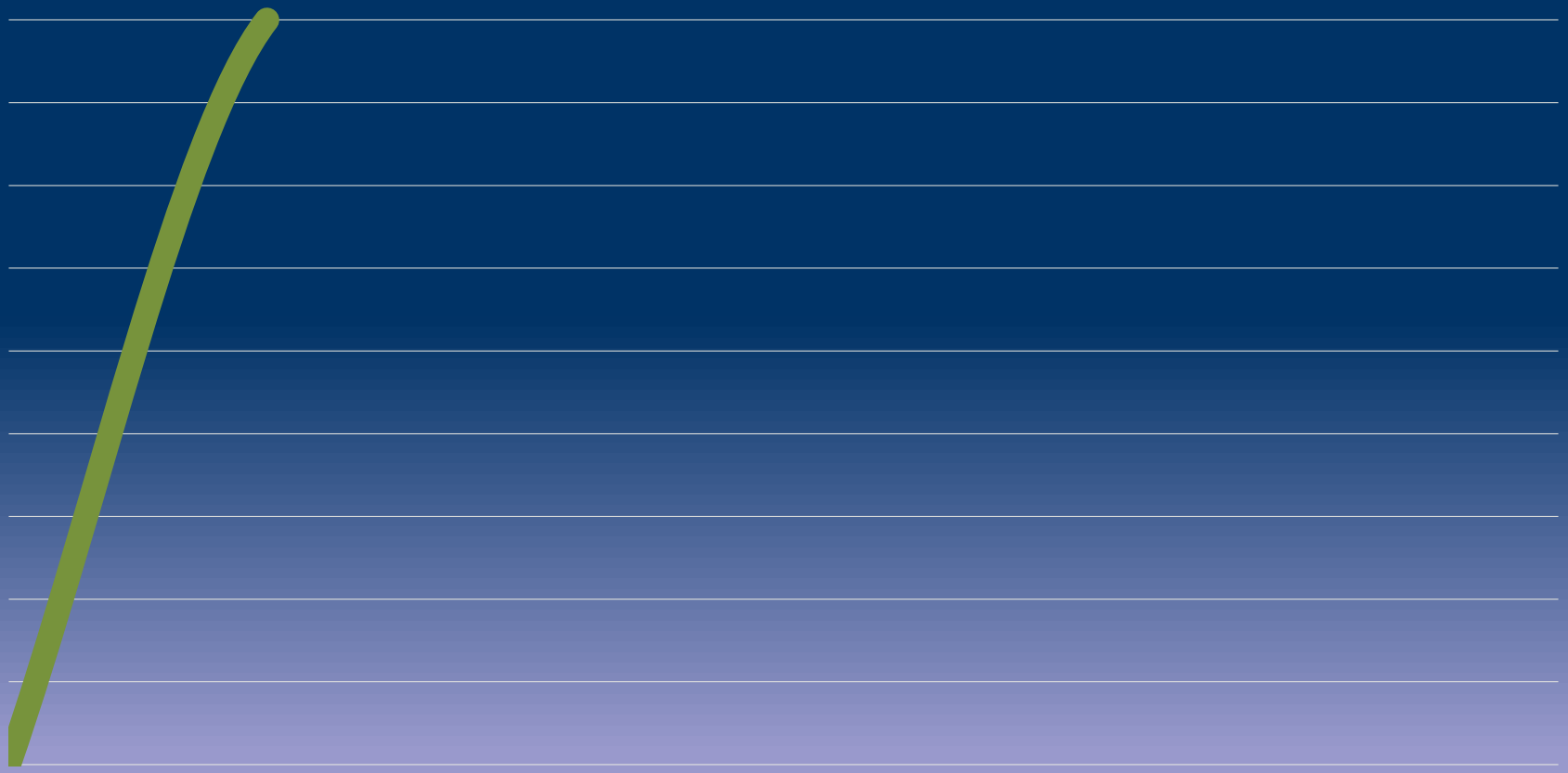
- Mythen & Fakten
- Praxisbeispiel
- Ressourcen

MYTHEN & FAKTEN

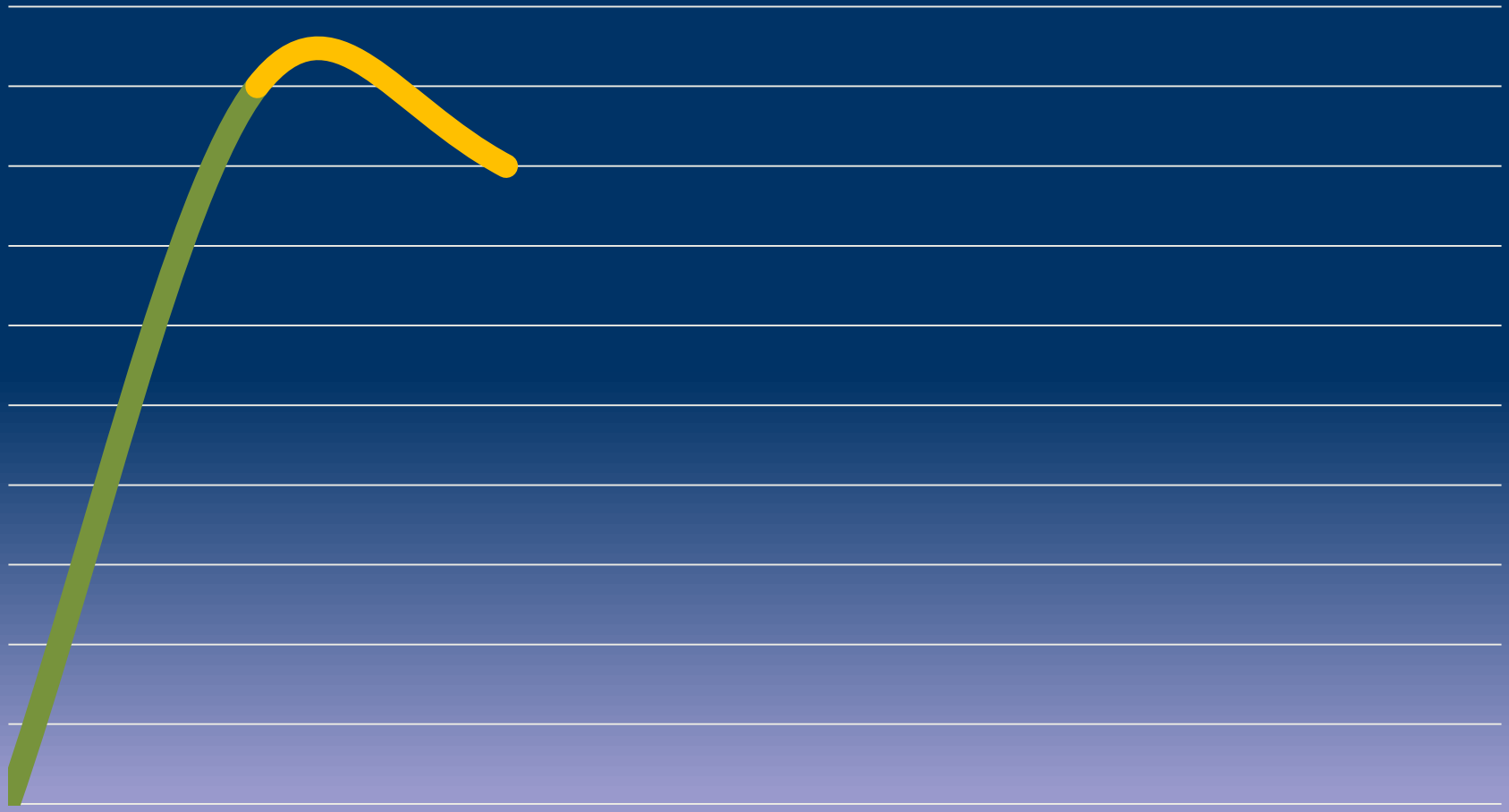
Mythen

- Refactoring ist unnötig da der «Big Rewrite» unsere Probleme lösen wird
- Wenn es nicht defekt ist macht es keinen Sinn es zu reparieren
- Planlos entwickeln ist kein Problem da Refactoring meinen Code verbessern wird
- Refactoring geht auch ohne Tests
- Refactoring ist immer angebracht

Motivationskurve

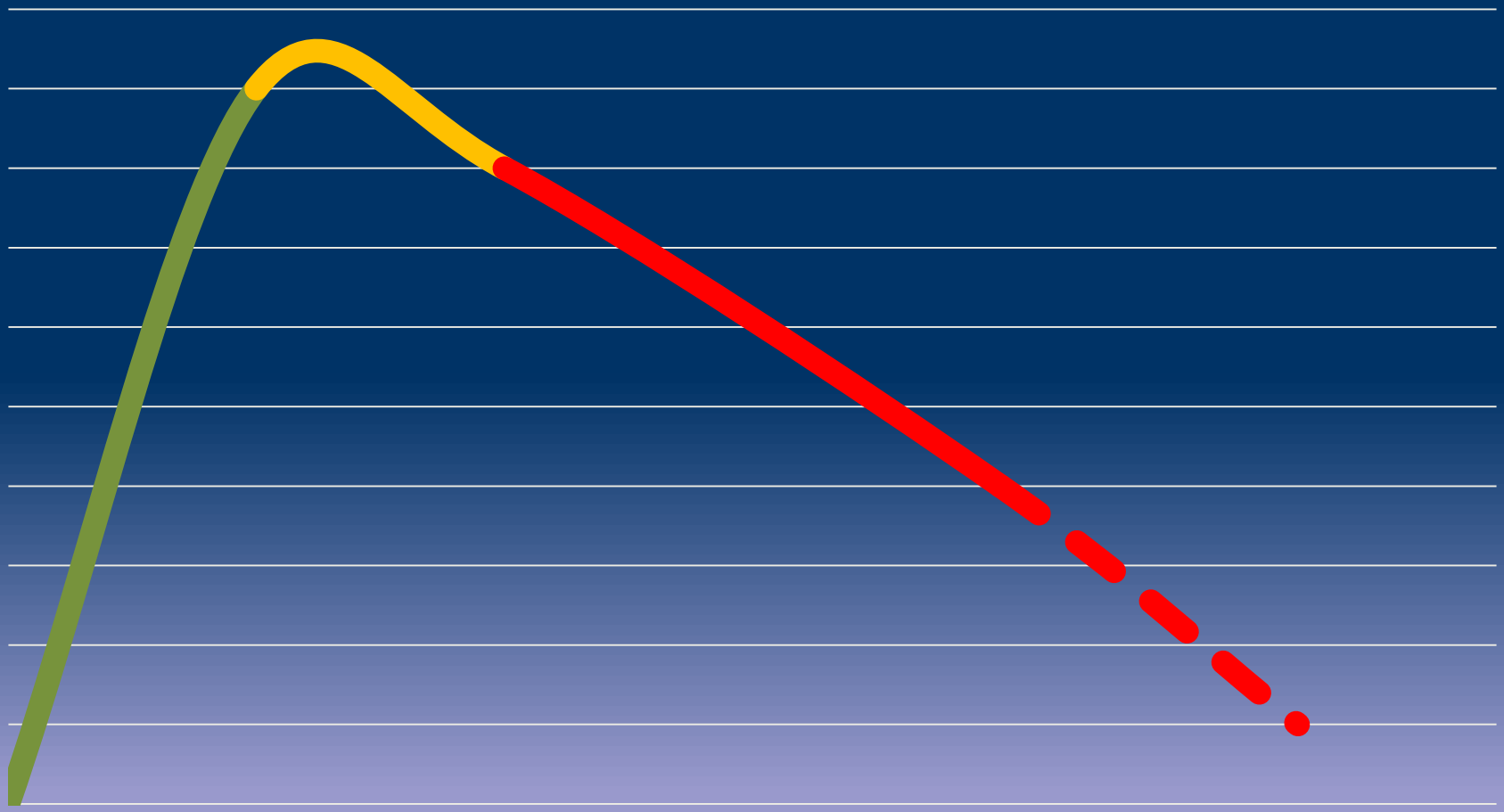


Beta Version

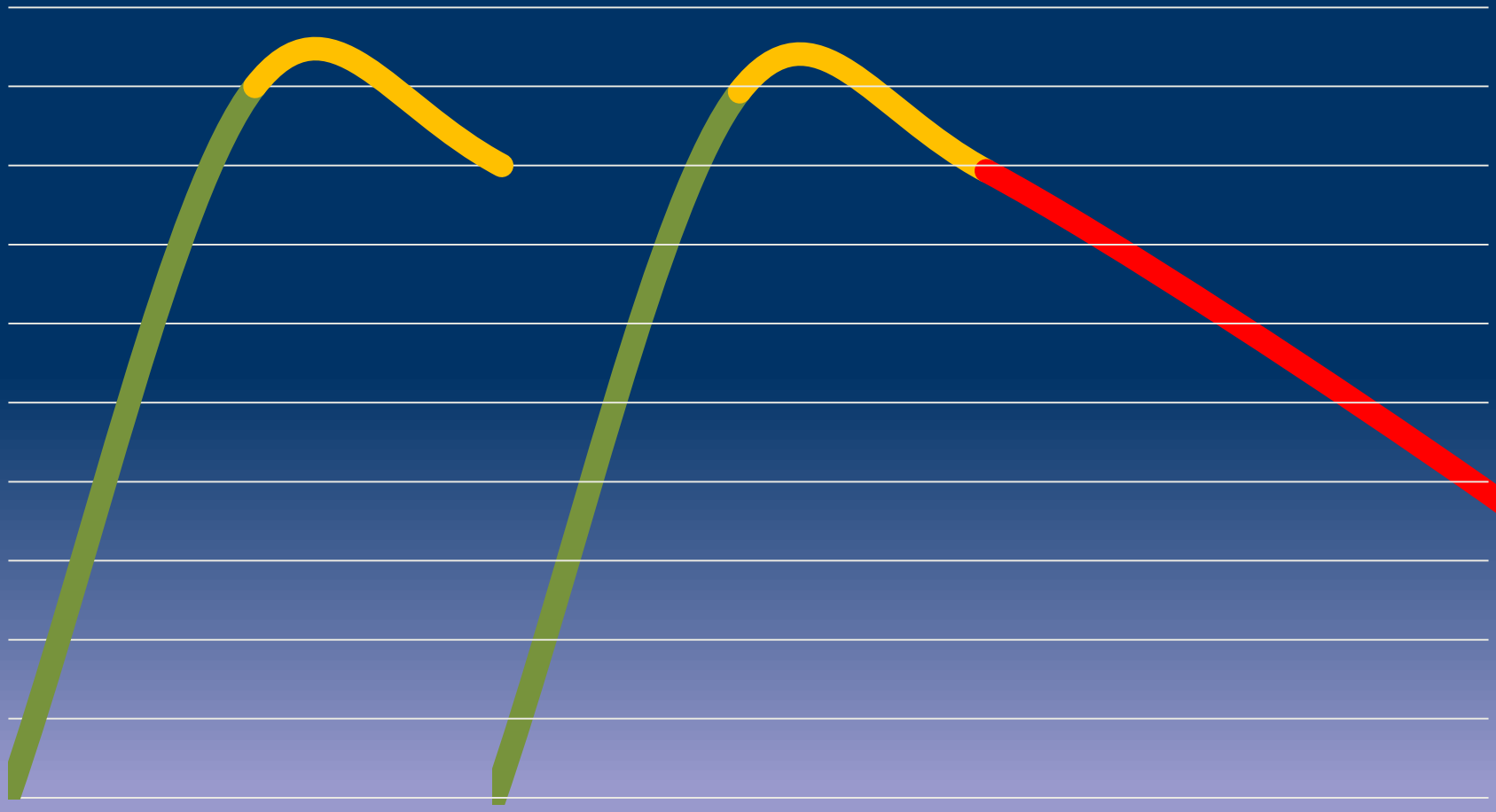


Version 1

Version n

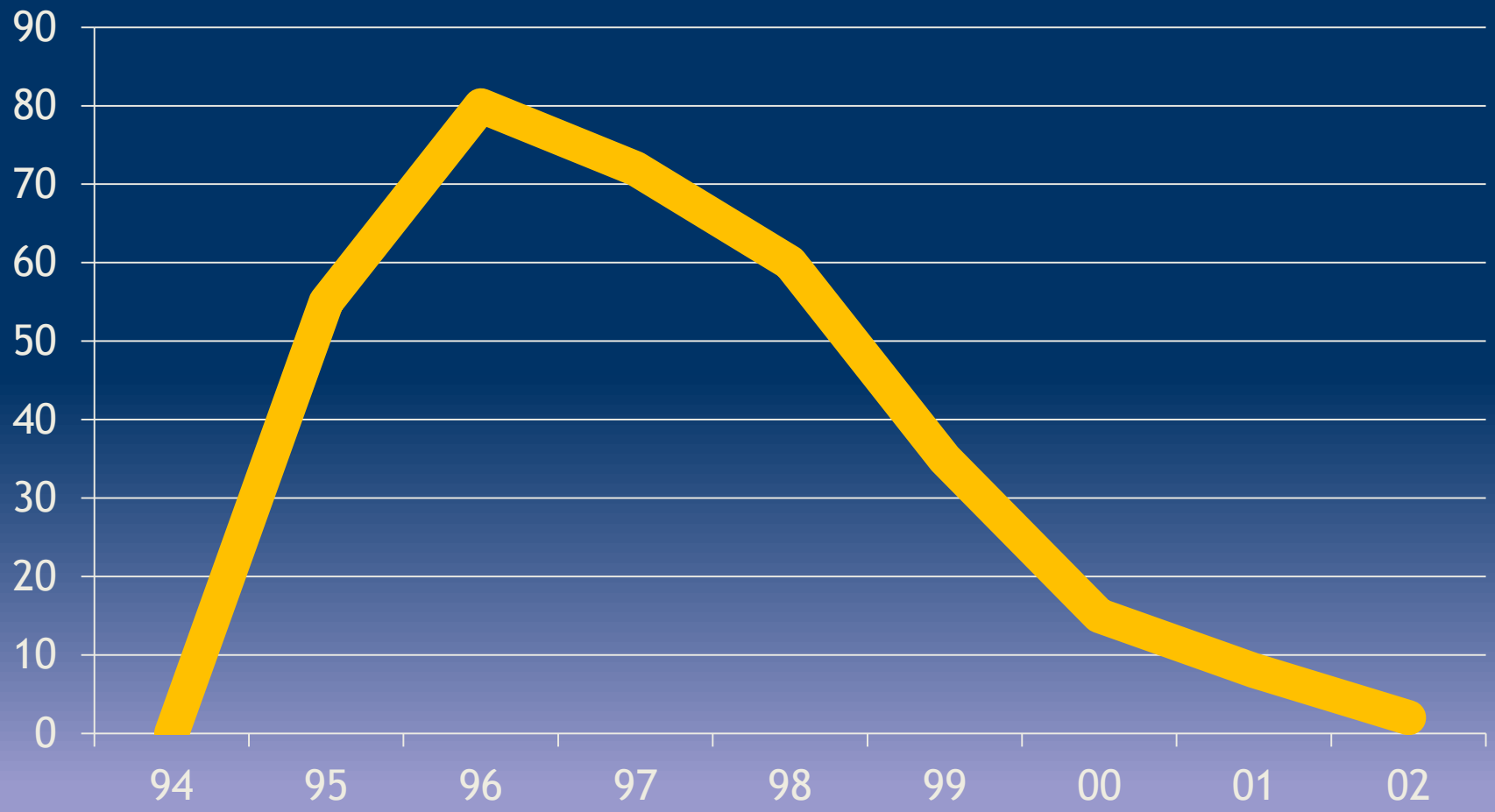


Neustart?



Netscape

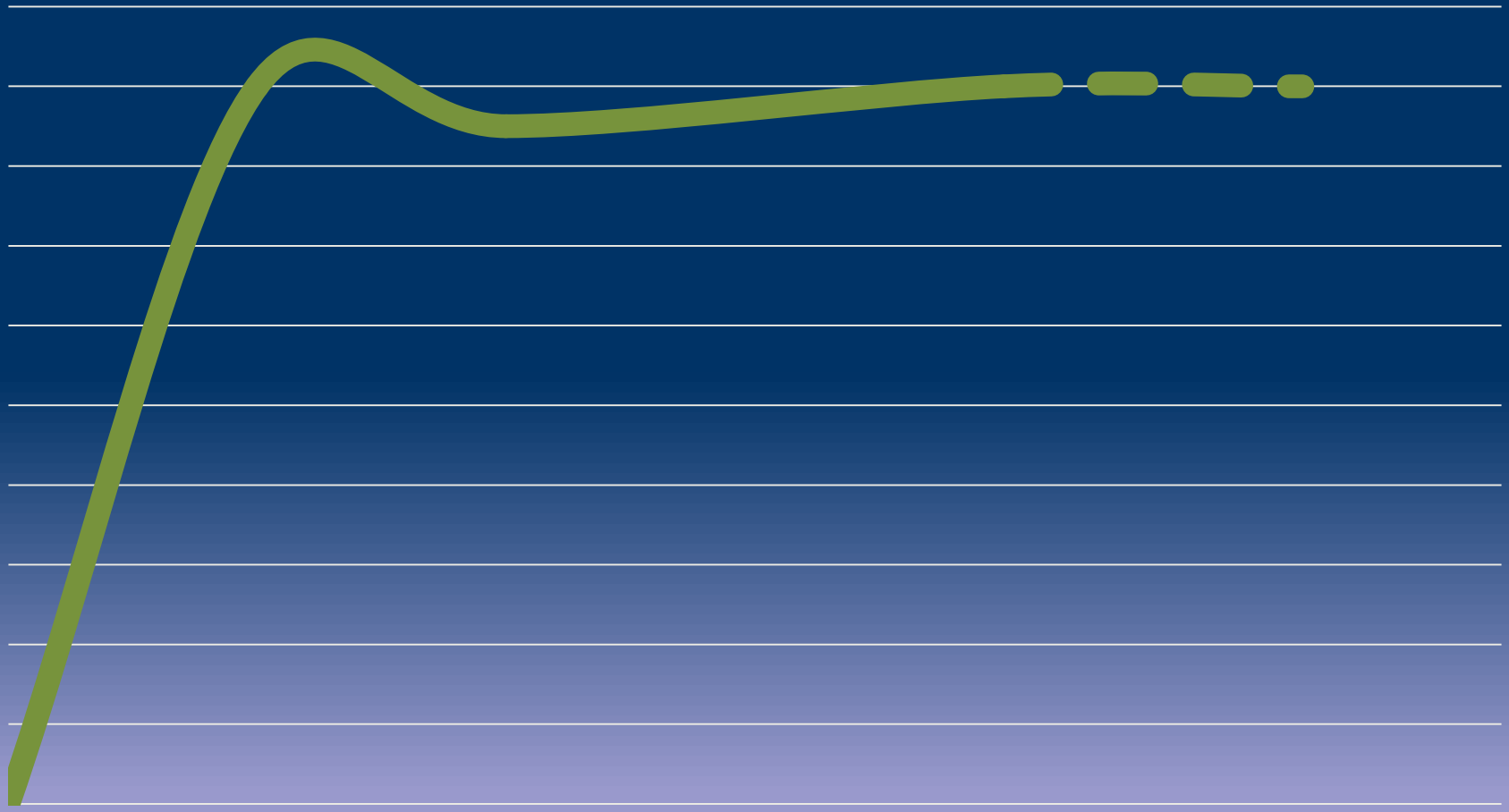
Erfolg des «Big Rewrite»



Big Rewrite

- Schlagen meistens fehl
- Der Kunde will nicht warten
  - Auf Funktionalität die er bereits hat
- Probleme werden sich wiederholen
- Aufwand wird unterschätzt
- Kosten explodieren

Wunsch



Refactoring

- Beibehalten der Funktionalität
- Korrektur der Struktur
- Kann Probleme lösen durch
  - Bessere Namensgebung
  - Reduktion der Komplexität
  - Wiederverwendung von Code

Refactoring Katalog

- Von Martin Fowler
- >45 Refactorings
  http://www.refactoring.com/catalog/
- Definiert eine gemeinsame Sprache

Oft verwendete Refactorings

- Rename Method
- Extract Method
- Move Method
- Extract Interface
- Form Template Method
- Reverse Conditional
- Introduce Parameter Object

Wann?

- Funktionalität soll beibehalten werden
- Technologie bleibt gleich
- Weiterentwicklung geplant
- Schrittweises vorgehen möglich

Wann nicht?

- Aktuelle Funktionalität passt nicht
- Technologiewechsel vorgesehen
- End-of-Life absehbar
- Keine Ressourcen vorhanden
- Ziel unbekannt
- Qualität egal

Kein Refactoring, wenn…

- … keine Tests vorhanden sind
- … Funktionalität verändert wird
- … einfach etwas ausprobiert werden soll

PRAXISBEISPIEL

Vorbereitung

- Ziel klären
- Umfang reduzieren
- Tests schreiben
  - TDD oder Test-First: Direkt zu Refactoring

Characterization Test

- Dient zum Verstehen von Code
- Ermöglicht Funktionserhalt
- Warnt vor unbeabsichtigten Änderungen
- Startet bei bestehendem Code
  - Geht auch mit Legacy Code

Ausgangslage

```
1   using System;
2   using System.Linq;
3
4   namespace RefactoringDemo
5   {
6       public class Conventions
7       {
8           public string CreateName(string input)
9           {
10              Validate input
28
29              Preprocessing
42
43              Handle case input too long
71
72              Handle case input too short
100
101             Create new name
117         }
118
119         other methods...
185     }
186 }
187
```

- Alles 1 Methode
- Viele Regions
- Keine Kommentare

1. Test

```
[TestClass]
public class CharacterizationTest
{
    [TestMethod]
    public void TestA()
    {
        var sut = new Conventions();
        var result = sut.CreateName("a");
        Assert.AreEqual("", result);
    }
}
```

```
Assert.AreEqual failed.
Expected:<>. Actual:<a>.
```

1. Test korrigiert

```
[TestClass]
public class CharacterizationTest
{
    [TestMethod]
    public void TestA()
    {
        var sut = new Conventions();
        var result = sut.CreateName("a");
        Assert.AreEqual("a", result);
    }
}
```

Success

2. Test

```
[TestMethod]
public void TestA()...

[TestMethod]
public void TestB()
{
    var sut = new Conventions();
    var result = sut.CreateName("aB");
    Assert.AreEqual("", result);
}
```

```
Assert.AreEqual failed.
Expected:<>. Actual:<Ba>.
```

2. Test korrigiert

```
[TestMethod]
public void TestA()...

[TestMethod]
public void TestB()
{
    var sut = new Conventions();
    var result = sut.CreateName("aB");
    Assert.AreEqual("Ba", result);
}
```

Success

Weitere Tests

```
[TestClass]
public class CharacterizationTest
{
    [TestMethod]
    public void TestA()...

    [TestMethod]
    public void TestB()...

    [TestMethod]
    [ExpectedException(typeof(ArgumentException))]
    public void TestC()...

    [TestMethod]
    [ExpectedException(typeof(ArgumentException))]
    public void TestD()...

    [TestMethod]
    public void TestE()...
}
```

Wie weiter mit den Tests?

- Behalten?
  - Bessere Namen nötig
  - Refactoring der Tests
  - Commit wenn OK
- Wegwerfen?
  - Tests bleiben wie sie sind

Refactoring der Tests

- Rename Method
- Extract Method
  - Statt Code zu kopieren
- Wirklich nur Tests verändern!

Refactoring der Tests

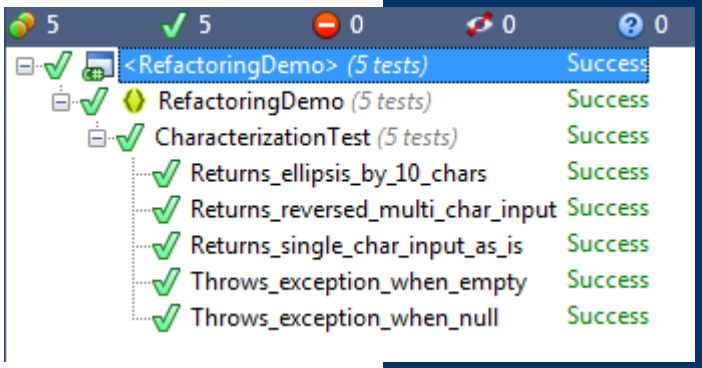
```
[TestClass]
public class CharacterizationTest
{
    [TestMethod]
    public void Returns_single_char_input_as_is()...

    [TestMethod]
    public void Returns_reversed_multi_char_input()...

    [TestMethod]
    [ExpectedException(typeof(ArgumentException))]
    public void Throws_exception_when_empty()...

    [TestMethod]
    [ExpectedException(typeof(ArgumentException))]
    public void Throws_exception_when_null()...

    [TestMethod]
    public void Returns_ellipsis_by_10_chars()...
}
```

Refactoring der Methode

- Namensgebung überprüfen
  - Methode
  - Parameter
  - Variablen
- Extract Method
- Auskommentierten Code löschen

Refactoring der Methode

```
1   using System;
2   using System.Linq;
3
4   namespace RefactoringDemo
5   {
6       public class Conventions
7       {
8           public string CreateName(string input)
9           {
10              if (String.IsNullOrEmpty(input))
11                  throw new ArgumentException("input must be set");
12
13              if (input.Length >= 10)
14              {
15                  return CreateEllipsis(input);
16              }
17
18              return ReverseString(input);
19          }
20
21          private static string ReverseString(string input)...
27
28          private static string CreateEllipsis(string input)...
32
33          other methods...
99      }
100 }
```



Bugfixes nötig?

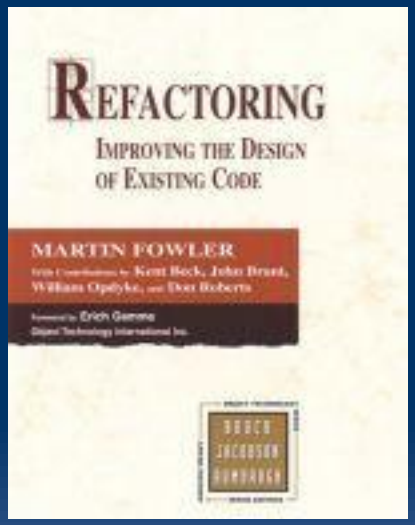
- Erst Refactoring abschliessen
- Tests laufen lassen
- Commit wenn alles OK
- Dann erst Bugfixes beginnen

Wozu dies alles?

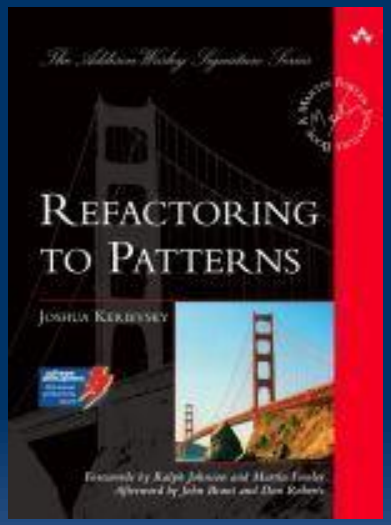
- Kleinere Methoden
  - sind einfacher zu verstehen
  - lassen sich einfacher wiederverwenden
- Tests
  - belegen den Funktionserhalt
- Basis für weitere Verbesserungen
- Refactoring ist der Weg, nicht das Ziel

RESSOURCEN

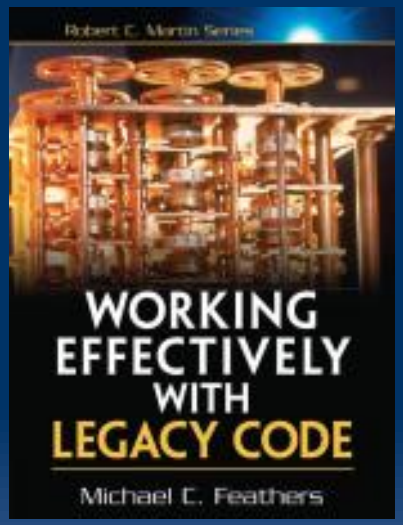
Bücher



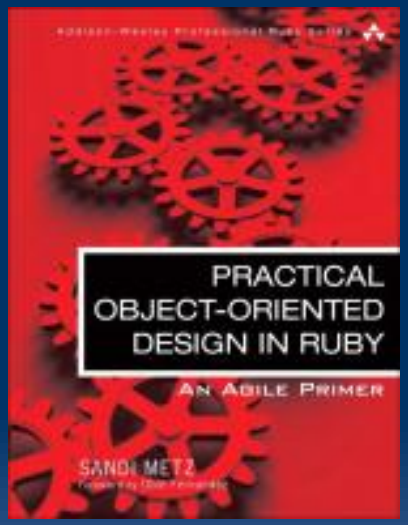
goo.gl/fPtvT



goo.gl/dCgsS



goo.gl/NkcDM



goo.gl/VkgbI

Videos

- Therapeutic Refactoring (Katrina Owen) http://goo.gl/JUKsm
- Go Ahead, Make a Mess (Sandi Metz) http://goo.gl/10ybu

Tools

- Können einem viele Arbeit abnehmen
- ReSharper:
  http://www.jetbrains.com/resharper/
- Moq:
  https://github.com/Moq/moq
- TypeMock:
  http://www.typemock.com/

Fragen und Diskussion