Refactoring Legacy Code

Dmytro Mindra
CTO, Opower Ukraine
Odessa, Ukraine
"A true legacy gets better with age."

---Uncle Bob
Dmytro Mindra

CTO, OPOWER
Microsoft MVP

http://www.infoq.com/author/Dmytro-Mindra

http://blogs.unity3d.com/ru/author/dmitriy/

2010 - Lohika
2012 - Microsoft
2013 - Unity Technologies
2015 - Opower

@dmytromindra
What do you call “legacy code”?

1. Code without tests
2. Code without specification
3. Code written long time ago.
4. Code that is hard to understand.

MakeFlagWavingBastardWaveHisFlagWhichIsTheProbablyTheLastThingHeWillEverDo()

Found in Carmageddon 1 debugging symbols dumped

#BuildStuffLT http://buildstuff.lt @dymtromindra
The Golden rule of software development

• It works - don’t touch it!
Legacy code – any code that you are afraid to change.
/*
When I wrote this, only God and I understood what I was doing
*/

// Now, God only knows
War Stories

...and that one is for the time I got 3 stars on every single level of Angry Birds.
Zero LOC (won’t get better than this)

"Hey, I’ll start with a simple prototype"

Supervisor wanted to see a demo

"Let’s keep it a prototype"

Spoke to developer friend and heard about software testing for the first time

Guilt

Deadline

© 2012 LALITH SURESH
Developers are born Brave
It’s hard to restrain code complexity growth when code evolves.

https://www.youtube.com/watch?v=w1q-vtcYYWE
Microservices

Monolithic/Layered

Micro Services
Disposable code
Notifications & logging built in
How do I get there?
"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure." -- Martin Fowler

What do developers want?

REFACTOR

ALL THE THINGS
Are you going to refactor with good intention(goal) in your mind?

Bug Fixing - 5.0 and 5.1
Fixing Disabled Tests
Writing new Native Tests
Refactoring

Is your goal measurable?
Can you prove that code is better now?
A GOOD ARCHITECT LEAVES A FOOTPRINT

WE GAVE UP FINDING PROPER NAMES FOR THE LAYERS LONG AGO. SINCE THEN WE JUST NAME THEM AFTER THEIR ARCHITECTS.

John
Judy
Marc
David
Transport
Transformation
Business Rules
Business Processes
Business Logic
Persistence

DB
What is good?
What are the Values?

• Features
• Design
• Feedback

http://www.jbrains.ca/permalink/the-three-values-of-software
#BuildStuffLT http://buildstuff.lt @dmytromindra
<table>
<thead>
<tr>
<th></th>
<th>Make it Fast</th>
<th>Refactoring</th>
<th>Big Design Upfront</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Design</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Feedback</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Good Refactoing

• Done regularly
• Done in small steps
• Brings measurable value
• Is always measured and quantified
Why do we change the code?

• To add a feature
• To fix a bug
• To increase performance

We change the code with some concrete and measurable goal in mind!
The most popular technique to deal with Legacy Code

Edit and pray (JDD)
A much better option

Cover and Modify
Getting Started

• Identify change points
• Find test points
• Break dependencies
• Write tests
• Make changes
Where to start?

• Identify change points
  – Legacy code is usually hard to understand
  – Change points are usually highly distributed
Getting Started

• Identify change points
• Find test points
  – We have to be sure that the behavior of the system has not changed
Make Sure that your changes are “safe”
ApprovalTests
// Magic. Do not touch.
Integration Tests

• They are good while you are changing the code.

• Don’t forget to get rid of them later.
Getting Started

- Identify change points
- Find test points
- Break dependencies
  - Can we instantiate/use class in isolation?
  - Can we get computation results?
Sensing and Separation

• We break dependencies to **sense** when we can't access values our code computes

• We break dependencies to **separate** when we can't even get a piece of code into a test harness to run
Seams

- **Seam**: a place where you can alter behavior of your program without editing in that place

- Preprocessor seams
- Link seam
- Object seam

http://blog.petterritchie.com/?p=2211
#BuildStuffLT  http://buildstuff.lt  @dmytromindra

Build Stuff LT 15  OPWWR
SOFTWARE DEVELOPMENT CONFERENCE
Separation

```java
public BarrierAssessment(int id, Client client, int homephone, int mobile,
int email, int transport, int licence, bool notInWorkForce,
bool unpaidWork, bool cdeProgram, bool notWorkingBut,
int paidWork, string employerOcc, string reason,
int highlevel, string further, int stableAccomodation,
int loneParent, int livingWith, int policeProblems,
int courtPending, int preventFrom, int medical,
int drugTest, int currentBusy, string scurrentBusy,
string reasonForNotGettingJob, int startTomorrow,
string jobGoals, int currentResume, int canvasLetter,
int referees, int interviewClothing, string otherBarriers,
int ec, LastModified lastModified, Date datestamp,
Date bupdated, string disabilityBarriers, Date licenceDueToExpire,
Date licenceDueBack, string currentSkills, string possibleSkills) {
```

In this case we can’t create instance of the class in isolation.
Separation

```java
public class Alarm {
    Sensor _sensor = new Sensor();

    public Alarm() {
    }
}
```
Extract Interface

Name “Sensor” is too generic in this case

```
PressureSensor _pressureSensor = new PressureSensor();
```

```
public Alarm()
{
}
```

Let’s extract “Sensor” interface

```
public interface Sensor
{
    double PopNextPressurePsiValue();
}
```
Parameterize Constructor

Sensor _sensor;

public Alarm()
{
    _sensor = new PressureSensor();
}

public Alarm(Sensor sensor)
{
    _sensor = sensor;
}

Default constructor still creates dependency, but we can use “parameterized constructor” to substitute it.
Extract and Override (step 1)
Factory Method

Old code:
```java
public void Check()
{
    double psiPressureValue = _sensor.PopNextPressure PsiValue();
}
```

New code:
```java
protected virtual Sensor GetSensor()
{
    return _sensor;
}

public void Check()
{
    double psiPressureValue = GetSensor().PopNextPressure PsiValue();
```
Extract and Override (step 2)

Factory Method

```java
public class TestableAlarm : Alarm
{
    protected override Sensor GetSensor()
    {
        return new TestableSensor();
    }
}
```
Extract and Override

3rd party

```csharp
parms.Add(conn.CreateParameter(FormatParamName("role")),
parms.Add(conn.CreateParameter(FormatParamName("name")),

using (var rdr = cmd.ExecuteReader())

Extract

protected virtual DbDataReader ExecuteReader(DbCommand cmd)
{
    return cmd.ExecuteReader();
}

Override

protected override DbDataReader ExecuteReader(DbCommand cmd)
{
    return new DataTableReaderStub();
}
```

#BuildStuffLT http://buildstuff.lt @dmytromindra
Sensing with Extract and Override

```csharp
parms.Add(conn.CreateParameter(FormatParamName("uID"), userId));
parms.Add(conn.CreateParameter(FormatParamName("rID"), roleId));

cmd.ExecuteNonQuery();

protected virtual void ExecuteNonQuery(DbCommand cmd)
{
    cmd.ExecuteNonQuery();
}

public class TestableDbRoleProvider : DbRoleProvider
{
    public string CommandText;

    protected override void ExecuteNonQuery(DbCommand cmd)
    {
        CommandText = cmd.CommandText;
    }
}
```
Getting Started

• Identify change points
• Find test points
• Break dependencies
• Write tests
  – Slow and complex tests are still fine here
  – Our goal is safety
  – Whatever guarantees safety is ok
Getting Started

• Identify change points
• Find test points
• Break dependencies
• Write tests
• Make changes in small steps and **refactor** while changing.
Little Summary

Refactoring driven by value
Micro Services
Disposable code
Measured code quality
Refactoring for testability
Safe refactoring techniques
... and
Refactoring is like sex ;)

• One mistake and you have to support it for the rest of your life.

• Once you get started, you’ll only stop because you’re exhausted.

• You can do it for money or for fun.
Refactoring is like sex ;)

• It’s not really an appropriate topic for dinner conversation.

• One little thing going wrong can ruin everything.

• Sometimes it’s fun to use expensive toys.
// I dedicate all this code, all my work, to my wife, Darlene, who will // have to support me and our three children and the dog once it gets // released into the public.
REMEMBER: Working with Legacy Code is all about “Safety”
// Questions?
References
Working Effectively With Legacy Code

Michael C. Feathers

#BuildStuffLT  http://buildstuff.lt  @dmytromindra
Your Code as a Crime Scene

Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs

Adam Tornhill
Exercise

https://github.com/emilybache/GildedRose-Refactoring-Kata
References

Refactoring Value or Waste

Code Quality Assurance
References


• http://www.slideshare.net/yodamann/icsa-2014-mudsustaining-architecture
References

• http://claysnow.co.uk/the-beer-belly-testing-anti-pattern/

• http://www.slideshare.net/excellaco/working-effectively-with-legacy-code-15594623

• http://joel.franusic.com/2015/02/27/notes-on-art-and-fear/
References


http://www.slideshare.net/kaunasjug/bye-bye-cowboy-coder-days

http://www.slideshare.net/nashjain/working-effectively-with-legacy-code-presentation

http://minds.coremedia.com/2013/08/21/sokahh-coding-dojo-legacy-code-kata/