Sustainable Legacy Code Refactoring: 
A Systematic and Stepwise Approach

Amr Noaman Abdel-Hamid

Agile Academy

Agile2014, Orlando
About Amr

• Agile practitioner, coach, trainer, & consultant
• Agile Academy co-founder
• SCEA, CSP, 6σ black-belt for IT
• Co-Author of the ‘Process Increments’ method, for software process improvement
• Initiator of Egypt’s GoAgile program at 2011, an Government initiative intended to boost lean & agile software development in Egypt
A Story of a Great Team
What's in common?

What's different?

What's in common?
Agenda

- Challenges
- Patterns of Legacy Code Refactoring
- What makes it fail

Roadmap of Sustainable Refactoring

- Quick-wins
- Divide and Conquer
- Inject Quality In
Challenges

- Frequent changes
- Unreliable code
- Shorter timescale

→ Immature attempts to refactor
Patterns for Legacy Code Refactoring

• Whole-or-nothing
• As-per-the-book
• Technical hero
• Try-retry
Does It Work This Way?
It doesn’t Work!
Failed to automate tests

Vague & hazy objectives

Technical Glut Trap

It’s none of the managers’ business!

Unsustainable Development Pace
Refactor or Automate Tests?

Which comes first?
We are busy refactoring the product code for the benefit of all of us.

It is a highly technical stuff which you will not grasp, even if you tried hard.

We need to concentrate. Please do not keep nagging for status and end dates.

When we finish, we’ll let you know.

Thank you for your understanding.
Technical Glut Trap
Why not fix this issue as well.

Hmmm, what about applying this pattern. It may affect only several other modules.

We’d better refactor this module, it’s just the right time.
Technical Glut Trap

no control over technical changes in the code

Code changed

+ Ideas to change the code

Changed applied
How to become unsustainable with refactoring?
Mainline (Trunk)

Project A

RC1

Project B

R1

Sub-task of project B

Refactoring Branch
How to Become **More** Unsustainable with Refactoring?

*Work overtime when tired and produce mistakes **10 times** more than normal!*  

Ref: *Awake is the next sleep* (documentary)

---

**Unsustainable development**

More bugs due to fatigue  
More time on development
Vague & hazy objectives

Failed to automate tests

Technical Glut Trap

It’s none of the managers’ business!

Unsustainable Development Pace
Agenda

✓ Challenges
✓ Patterns of Legacy Code Refactoring
✓ What makes it fail

Roadmap of Sustainable Refactoring

• Quick-wins
• Divide and Conquer
• Inject Quality In
Quick Wins
- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer
- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In
- Cover components with automated tests
- Enhance components internal design

Continuous Review
Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
What’s Evil about Dead Code?

Out of the $100 Billion spent on software in the 90’s, $70 Billion are spent on Maintenance

Cost of locating bugs

60%

40%

Everything else!

Fixing
Testing
Reviews
Integration
Integration & system testing
Deployment
User Acceptance

....
What’s Evil About Dead Code?

- Creates noise when locating bugs
- Extremely easy to pinpoint and remove:
  - 4-7% of the code is dead
  - Removing dead code removed 10% of duplicate code as well!
  - Removing dead code takes 1-3 days

Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
Code Duplicates (Clones)

System Code

40% Code clones

52% Inconsistent clones

18% faults in system

Overall, 20% effort increase in maintenance due to code clones

Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
Eclipse 3.0 Token Analysis

Ref: http://www4.informatik.tu-muenchen.de/publ/papers/deissenboeck_pizka_identifier_naming.pdf
Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

This Stuff is Risky Isn’t It?

Divide & Conquer

- Cover & encapsulate components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
No, It’s Not!

<table>
<thead>
<tr>
<th>Metric</th>
<th>Release 5.5 (Before refactoring)</th>
<th>Release 5.6 (During Refactoring)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total bugs detected</td>
<td>128</td>
<td>176</td>
</tr>
<tr>
<td>% of Regression bugs</td>
<td>29.7%</td>
<td>25.1%</td>
</tr>
<tr>
<td>Average bug fixing cost (hours)</td>
<td>1.97</td>
<td>1.8</td>
</tr>
</tbody>
</table>

% New Bugs

% Regression Bugs
Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

How is this work Tacked?

Divide & Conquer

- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
Tracking Code Size Reduction

Code Size Reduction Target (CSR Target) =

- 100% of Dead Code
- 90% of Exact Clones
- 60% of Similar Clones

![Graph showing iterations and corresponding KLOC values]
Size Reduction Speed Run Chart

- Number of LOC removed per hour of work
- Calculated every iteration
Number of Violations burn-down

Number of methods with LOC > 10
Use them to involve busy managers and gain their support.
Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
Before quick wins

After quick wins
How?

• Group “similar” code together

• Similar code may be:
  – Functional
  – Utility
  – Port
  – Architectural

More discussion about types of components at:
‘Divide & Conquer’ Using ConQAT
‘Divide & Conquer’ Using ConQAT

This should be the architecture

This is the really happens!
Initial Architectural Analysis
Useful Metric at This Stage

- Number of Violations (Target 0)
- Class/component Coupling (Target 0)
Quick Wins
- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer
- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In
- Cover components with automated tests
- Enhance components internal design
Wrap Components with Automated Tests
Useful Metric at This Stage

• Test Coverage:

\[ \text{target test coverage} = \frac{\text{included LOC}}{\text{total LOC}} \% \]

Where excluded LOC is non-business logic code like:
- GUI code
- Auto-generated code
- Third party code
Target test coverage = 58% (103k / 176k)
Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review
Continuous Review

• Stand away from the team!
• Monitor the code quality
• Use CI tools
Quick Wins
- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

Divide & Conquer
- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In
- Cover components with automated tests
- Enhance components internal design

Continuous Review

Questions?
Keep In Touch!

amr@agileacademy.co
amr.noaman@gmail.com

www.agileacademy.co
Blog: amr.agileegypt.org
@amrnoaman