Effects of Dependency Injection on Maintainability

Kate Razina
Overview

• Introduction
  - Maintainability
  - Dependency Injection
• Hypothesis
• Research
  - Measuring Maintainability
  - Data Collection
• Results
• Conclusion
  - Future Work
Maintainability

- **Definition**
  
  The ease with which a software system or component can be modified. Modifications may include extensions, porting to different computing systems or improvements\(^1\)

- **Maintainability**
  
  - Most time consuming part of software life cycle
  - 65% to 75% of total time\(^2\)

Dependency Injection

No Dependency Injection

Dependency Injection
Dependency Injection

No Dependency Injection

Dependency Injection
Dependency Injection (IOC)

- **Types**
  - Constructor
  - Setter

- **Used in**
  - Spring framework
Dependency Injection XML File (Constructor Injection)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="activityHW" class="student.DoingHw"/>
  <bean id="student" class = student.MastersStudent">
    <constructor-arg>
      <ref bean="activityHW"/>
    </constructor-arg>
  </bean>
</beans>
```
public class MastersStudent implements Student{
    private Activity activity;
    public MastersStudent(Activity activity){
        this.activity = activity;
    }
}
Why Use Dependency Injection?

- Container is provided, configured with an XML file
- A MastersStudent can do any activity with a slight change of the XML file
- We can test MastersStudent without subsequently testing the implementation of Activity
- Responsibility for coordination collaboration between objects is transferred away from the objects themselves
Why Use Dependency Injection?

- Coupling between objects is reduced or is it?
  - Decoupled, testable, easy to maintain
- Couplings become more configurable and flexible.

Hypothesis

The pattern of dependency injection significantly reduces dependencies between classes in a piece of software, therefore making the software more maintainable.
How Do We Measure Maintainability?

- Single Metric Measures
  - Cohesion (LCOM)
  - Coupling
    - CBO
    - 3
  - RFC
    - 4

```
MastersStudent
  Interface
  Activity
  Implementation
  DoingHW
  Implementation
  PlayingSoccer
```

```
MastersStudent
  setActivity{
    new DoingHW();
    new PlayingSoccer();
    new EatingDinner();
  ;}

Implementation
DoingHW

Implementation
PlayingSoccer

Implementation
EatingDinner
```
How Do We Measure Maintainability?

- Lower coupling
  - Improved maintainability
  - Decreases interdependencies between modules\(^4\)
  - The higher the coupling between modules, the more difficult the modules are to understand, change, and correct \(^5, 6\)

---


Data Collection

- Spring Framework
- Project Collection
  - Sourceforge.net
  - 40 projects
Data Collection

- CKJM Tool
  - Metrics
    - CBO, RFC, LCOM
    - Weighted Methods per Class (WMC)
    - Depth of Inheritance Tree (DIT)
    - Number of Children (NOC)

- Number of DIs Metric
  - Measure to what extent each Spring project uses dependency injection
Data Collection

- Analysis
  - 2 Sample t-test
    - p-value $\leq 0.05$
## Results

- **T-Test**

<table>
<thead>
<tr>
<th></th>
<th>CBO</th>
<th>RFC</th>
<th>WMC</th>
<th>DIT</th>
<th>LCOM</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Non-Spring</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Results

T-Test

Chart of CBO, RFC, WMC, DIT, LCOM, CE vs C1
No obvious correlation exists between the presence of dependency injection and the Chidamber and Kemmerer metrics.

Examine CBO & RFC results closely

- Correlation between lower average CBO/higher DI percentage and vise-versa
- Correlation between lower average RFC/higher DI percentage and visa-versa
Results

<table>
<thead>
<tr>
<th></th>
<th>CBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring</td>
<td>5</td>
</tr>
<tr>
<td>Non-Spring</td>
<td>6</td>
</tr>
</tbody>
</table>

![Graph showing lower CBO and higher DI](image)
Conclusion

- **Hypothesis:**
  - The pattern of dependency injection significantly reduces dependencies between classes in a piece of software, therefore making the software more maintainable.

- **Results:**
  - No obvious correlation exists between the presence of dependency injection and average CBO and RFC.
  - We cannot say that using dependency injection will lower coupling and therefore reduce maintainability.
  - Correlation between lower average CBO/higher DI percentage and vice-versa.
  - Correlation between lower average RFC/higher DI percentage and visa-versa.
Future Work

- Study coupling more
- Ideal percentage of DI
  - Burden of maintenance now includes xml code as well as java code
- DI / $\sum$CBO metric
Questions?