Recitation 13

Principles of Software

Brandon Rozek
rozekb@rpi.edu

Rensselaer Polytechnic Institute, Troy, NY, USA

April 2022

Outline

Two Things:

- Refactoring
- UI Design

Technical Debt

- *Technical debt* refers to the cost of picking a simpler solution as opposed to a maintainable one.
- As a project progresses, it may incur "interest" making it more difficult to implement changes.

Commonly caused by:

- Business Pressures
- Insufficient Specifications
- Tightly coupled components

Antipatterns

An *anti-pattern* is a common rseponse to a recurring problem that is usually ineffective or counterproductive.

Examples include:

- Big ball of mud: Lack of design
- God Class: One class handles everything.
- Poltergeists: Ephemeral controller classes that only exist to invoke other methods on classes.

Code Smells

Similar to antipatterns, a *code smell* indicates potential techincal debt within the codebase.

Five categories of code smells (by Mika Mäntylä):

- The Bloaters: Components too large to handle.
- The Object-Orientation Abusers: Not making use of OO features such as polymorphism.
- The Change Preventers: Hinders further development.
- The Dispensibles: Unnecessary components that should be removed.
- The Couplers: Captures high dependency between multiple classes.

Example Code Smells

The Bloaters:

- Large Class, Long Parameter List
- Data Clumps: Groups of variables passed around together in a program. It should be an object instead.

The Object-Orientation Abusers:

- Refused Bequest: An overriding of a method that is not a true function subtype.

The Change Preventers:

- Shotgun surgery : A single change requires changes to multiple classes.

Example Code Smells

The Dispensibles:

- Duplicated Code, Dead Code
- Lazy class / freeloader: a class that does too little.

The Couplers:

- Feature envy: a class that uses methods of another class excessively.
- Inappropriate intimacy: a class that has dependencies on implementation details of another class.

Static Code Analysis

Static code analysis is a methodology of analyzing code without executing it. Commonly called *linters*, these programs attempt to find bugs, stylistic errors, and code smells.

Examples for Java:

- Checkstyle: Style checking tool.
- Spotbugs: Static analysis tool for fidning bugs.

Refactoring

Refactoring is the process of restructuring code without changing its behavior.

There are two primary benefits:

- Maintainability: Make it easier to understand.
- Extensibility: Make it easier to modifier in the future.

Refactoring Techniques:

- Extract Method
- Move Method
- Replace temp with query
- Replace type code with State/Strategy

Refactoring Techniques

The three most common techniques are *abstraction*, *splitting*, and *relocation*.

Splitting involves the breaking apart of large classes or methods.

Relocation involves the renaming or moving of a component up the superclass or subclass.

Abstraction techniques include:

- Encapsulate Field (Getters/Setters)
- Generalize Type (Generics)
- Replace Type code with State/Strategy
- Replace conditional with Polymorphism

Extract Method

A splitting technique used to break up logical chunks into separate methods.

Approach:

1. Create a new method.
2. Move the extracted code to the new method.
3. Any referenced variables that are not declared in the extracted code, make into a parameter.
4. Return parameters that are modified and used after the extracted component.
5. Replace the extracted code with a method call.
6. Compile and test.

Move Method

A relocation technique used to decouple classes and move a method to a more appropriate location.

Approach:

1. Declare the method in the target class.
2. Appropriately copy the code from the source class to the target class.
3. Do take note of the (sub/super)-classes to capture all declarations of the method.
4. Turn the original method to a delegating method.
5. Compile and test.

Replace Temp with Query

A splitting technique that deals with side calculations and making it so that temporary variables are not inappropriately reused.

Approach:

1. Identify the temporary variable representing a side calculating.
2. Declare the type as `final`
3. Capture the calculation of that variable into a method (which we call query). Replace all occurrences of the side calculation with the query.
4. Note that the query method should be free of side effects.
5. Compile and test.

State Design Pattern

A *state design pattern* is commonly used to have behavior differ depending on some state.

Useful for when:

- Creating subclasses are not ideal.
- Objects may change between multiple states throughout runtime.

State Design Pattern Usage

Transform:

```
public class Person {
    private String name;
    private int mood;
}
```

To:

```
public class Person {
    private String name;
    private MoodType mood;
}
```

MoodType would then be an abstract class which is extended for each type of mood.

Commonly used with the "Replace conditional with Polymorphism" technique.

UI Design

Usability

Usability captures the capacity for a system to allow users to perform tasks *safely*, *effectively*, and *efficiently* while *enjoying* the experience.

Components of Usability

From Jakob Nielsen:

- Learnability: How easy is it to accomplish basic tasks on first encounter?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re-establish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they *recover* from the errors?
- Satisfaction: How pleasant is it to use the design?

Extra Components

The US government in addition to the ones in the last slide also include:

- Desirable: Image, identity, brand, and other design elements are used to evoke emotion and appreciation
- Findable: Content needs to be navigable and locatable onsite and offsite
- Accessible: Content needs to be accessible to people with disabilities
- Credible: Users must trust and believe what you tell them

Learnability & Memorability

Learnability:

- People normally do not learn a complete interface before using it.
- Aim for consistent design and behavior.

Memorability:

- Use common terminology. Avoid jargon.
- Use icons wisely

Errors

- Avoid mode errors.
- Use confirmation windows sparingly.
- Are errors few and recoverable?
- Add an undo option.

Fitt's Law

Fitt's Law is a predictive model of human movement used to model the act of pointing.

Intuitively the time it takes to move to the target area is a function of the ratio between the distance of the target and the width of the target.

Efficiency

- Try to minimize the number of clicks needed to perform an action.
- Avoid deep hierarchies.
- Make important targets big and nearby
- Provide shortcuts

Human Perception

- Response time of less than 100ms feels instantaneous
- 10fps is enough to perceive an image as moving.
- 8% of all males are red-green color blind.

Satisfaction

- Make system state visible
- Give prompt feedback (less than 100ms)
- Look "modern"

Prototyping

A *prototype* is a draft version that allows users to explore ideas and features before investing time and money into more permanent development.

- A *proof-of-principle prototype* serves to verify some key functional aspects of the intended design, but usually does not have all the functionality of the final product.
- A *working prototype* represents all or nearly all of the functionality of the final product.
- A *visual prototype* represents the size and appearance, but not the functionality, of the intended design.
- A *functional prototype* captures both function and appearance of the intended design, though it may be created with different techniques and even different scale from final design.

User Testing

- Start with a prototype
- Write up a few representative tasks
- Find a few representative users
- Watch them do tasks with the prototype

How to watch users

- Brief the user first
- Ask the user to think out loud
- Don't talk or make faces during their tasks.
- Take notes of any confusions or non-optimality.

Any Questions?