Announcements

- Assignment 3 questions
- Implemented new User Story as assigned includes finishing the confirmation and breaking into tasks
ANTI PATTERNS
“an anti-pattern is something that looks like a good idea, but which backfires badly when applied.”

–Jim Copliene
THE CODE IS MORE LIKE "GUIDELINES"
RATHER THAN ACTUAL RULES
AntiPattern Categories

- Software Architecture AntiPatterns
- Software Development AntiPatterns
- Project Management AntiPatterns
**AntiPatterns**

- A bad solution is always bad, a good solution could be one of many ways to solve a problem
- Less personal to criticize
- Shared Vocabulary
Software Architecture AntiPatterns
Big Ball of Mud

- Systems that are built up over time with no real architecture
- Is often time the quickest "solution"
- "If you don’t have time to do it right, you definitely don’t have time to do it twice"
- Solution?
God Class aka Blob

- Classes that have many responsibilities and many dependencies
- Solution is to break up class into smaller units
- Fixing can be very difficult
Lasagna Code

- A layer based architecture but there is no separation enforced on the layers.
- Layers need to be separate. There should be no calls from one layer to another.
One class with many different types of responsibility

Solution: Divide up functionality by responsibility

https://www.knife-depot.com/knife-434188.html
Stove Pipe Architecture

- A brittle grouping of poorly connected components
- Solution: Improve relationships between component

http://momerath-stock.deviantart.com/art/Rusty-Stovepipe-2-132395967
An analogy is used to describe the architecture. Instead of moving on to a more precise definition, the analogy is maintained even after it has lost its usefulness.

Justifications and architectural decisions may be based upon constraints imposed by an analogy instead of the problem at hand.
Politics Oriented Architecture

- Decisions made for political reasons, instead of for technological reasons
- Often political constraints are very real, as real as technical constraints
Software Development AntiPatterns
**Magic Numbers**

I literal number that appears in the code. e.g.

```plaintext
total = price * 1.075;
```

**VS**

```plaintext
CA_Sales_Tax_Rate = 1.075;
total = price * CA_Sales_Tax_Rate;
```
Spaghetti Code

✖ ...one of those things that "everybody else does."
✖ Code with very little structure. Any code can call or be dependent on any other code.
✖ A small change can have huge ripple effects
Cut-and-Paste Programming

- Cutting and Pasting code is very common, but it can create maintenance nightmares.
- This will lead to the same code in many different locations throughout the code.
- Potential anti pattern in Assignment 3
A single conceptual change that must be implemented in a large number of different locations in the code.

This is a sign that your code is not well organized, and will lead to maintenance problems down the road (if not right now)
Golden Hammer

✖ “When all you have is a hammer, everything looks like a nail”
✖ Something may be a good solution, but it should not be applied in every single situation.
Gold Plating

- “Perfect is the enemy of done”
- The point in a task when extra work adds little if any value to the project.
- Common trap for some personality types
Premature Optimization

- Optimizing before you have enough information to make educated conclusions about where and how to do the optimization.
- Bottlenecks can be difficult to predict. It’s best to wait till the optimization is needed.
Over Use Of Patterns

- Trying to apply every single pattern that is known to a single project regardless of if it applies or not.
- Adding too many patterns will inevitably result to too much complexity, with no benefit to the developers.
If It Aint Broke Dont Fix It

✖ Every developer should be at least attempt to improve every piece of code that they touch.
✖ Begin afraid to touch some code may be a sign of too few unit tests, poor design, or lack of understanding about the code.
 Dependency hell / DLL Hell

- When your software depends on a specific version of a DLL/Library, and that depends on another DLL/Library, and so on.
- A Single upgraded library can cause enormous problems
- Maven is a solution to this problem
Analysis Paralysis

✖ When a team spends too much time worrying about getting the perfect design, instead of starting to do something, which can later be improved on.
✖ Sometimes it's easy to lose sight of the real goal
Decisions being made without considering the entire system, then being passed on to another group/team with no thought for how to integrate/test/deploy. 
DevOps is an attempt to solve this problem
Every project is behind, so it is time to PANIC!
Panic becomes the status quo.
This can work in the short term, but not in the long term.
“After 5pm, you are writing tomorrow’s bugs”
Work/life balance should be sustainable
Measuring programming progress by lines of code is like measuring aircraft building progress by weight. –Bill Gates

You get what you reward.

Metrics make a poor way to determine programmer productivity
Software solutions that are “not invented here” are looked upon with suspicion, leading to a roll your own mentality for every single problem.

There should be a compelling reason not to use a tested, off the shelf component.
Scope Creep

✖ Scope Creep occurs when the requirements and scope for a project are not firmly controlled, leading to lots of small changes that eventually add up to large changes.
✖ It is important to stand up to this even as engineers
"Seagull managers fly in, make a lot of noise, dump on everyone, then fly out.” – Ken Blanchard
Don’t be this person.
Look out for this when applying for jobs.
Code Smells
"a code smell is a surface indication that usually corresponds to a deeper problem in the system”

–Martin Fowler
Code Smells

- An indication of a deeper problem
- Not an technically incorrect themselves
- Similar to AntiPatterns, but not not as objectively incorrect.
**Duplicate code**

- When there is duplicated code in various locations around the source code, that is a code smell
- Code can be char-for-char identical, token-for-token, or functionally identical
Code Smells:

- **Inappropriate intimacy**: A class that has dependencies on implementation details in another class.
- **Feature Envy**: A class that excessively uses the methods from another class.
- **Large Class**: A class that has grown too large.
Code Smells:

✖ Too many parameters: a long list of method parameters make the method hard to call, and can mean that the method is doing too much

✖ Long method: a method has become too large
Code Smells:

✖ **Inconstant Names:** if you have `Open()`, you should have a `Close()`

✖ **Uncommunicative Name:** is the name for a variable or method unclear?

✖ **Dead Code:** Ruthlessly delete code that isn’t being used. That’s why we have Versions Control Systems
TECHNICAL DEBT
Metaphor for how cutting corners can hurt

✖ Doing things “quick and dirty” causes technical debt to accrue.
✖ We can pay the interest or we can pay down the principle
✖ $200,000 Loan at 3.92% interest over 30 years = $350,427
Technical Debt

Reckless

“We don’t have time for design”

Prudent

“We must ship now and deal with consequences”

Deliberate

Inadvertent

“What’s Layering?”

“Now we know how we should have done it”

How to mitigate technical debt

- Define Technical Debt
- Raise Awareness
- Track your Technical Debt
- Make conscious decisions when to address it or not
- Use Best Practices for development
- Unit Tests
Refactoring
Types of Refactoring

- Composing methods
- Moving Features between Objects
- Organizing Data
- Simplifying Conditional Expressions
- Simplifying Method Calls
- Dealing with Generalization
Composing methods

- Extract Method
- Inline Method
- Inline Temp
- Replace Method with Method
- Object
Moving Features between Objects

- Move Method
- Move Field
- Extract Class
- Inline Class
- Remove Middle Man
Organizing Data

- Replace Data Value with Object
- Change Value to Reference
- Replace Array with Object
- Replace Magic Number with Symbolic Constant
- Encapsulate Field
- Replace Subclass with Fields
- Replace Type Code with Class
Simplifying Conditional Expressions

- Decompose Conditional
- Consolidate Conditional Expression
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
Simplifying Method Calls

- Rename Method
- Add Parameter
- Remove Parameter
- Parameterize Method
- Replace Parameter with Explicit Methods
- Replace Parameter with Method Call
- Replace Error Code with Exception
Dealing with Generalization

✖ Pull Up/Push Down Field
✖ Pull Up/Push Down Method
✖ Pull Up/Push Down Constructor Body
✖ Extract Interface
✖ Replace Inheritance with Delegation
Credits

Special thanks to all the people who made and released these awesome resources for free:

✗ Presentation template by SlidesCarnival
✗ Photographs by Unsplash