Welcome!

- We'll start in a moment :)
- We are NOT recording tonight's event. We may plan to take screenshots for social media.
    - ***If you want to remain anonymous***, change your name & keep video off.
- We'll introduce the hosts and break in-between for Q/A.
- We will make some time for Q&A at the end of the presentation as well.
- You can come prepared with questions. And, feel free to take notes.
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts.
    - Mute yourself when you aren't talking.
    - We want the session to be interactive.
    - Feel free to unmute and ask questions in the middle of the presentation.
    - Turn on your video if you feel comfortable.
    - Disclaimer: Speaker doesn't knows everything!

Check out:

- Technical Tracks and Digital Events
- Get updates – join the Digital mailing list
- Give us your feedback – take the Survey

WOMEN WHO CODE

WWCode Digital + **Backend**

Backend Study Group

June 17, 2021

WOMEN WHO CODE

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Inspiring women to excel in technology careers.
- Our vision: A world where women are representative as technical executives, founders, VCs, board members and software engineers.

Prachi Shah
Senior Software Engineer @ Metromile

- What is Backend Engineering?
- Software Design
- Design Patterns
- **Software Design Patterns [Part 4 of 5]**
  - Creational Design Patterns [4/22]
  - Structural Design Patterns [5/20]
  - Behavioral Design Patterns [6/3]
  - **Anti-patterns [6/17]**
  - Interview Questions/ Coding [7/1]

WOMEN WHO CODE

Backend Engineering

- What is Backend Engineering?
- Design, build and maintain server-side web applications.
- Concepts: Client-server architecture, API, micro-service, database engineering, distributed systems, storage, performance, deployment, availability, monitoring, etc.

Software Design

- Defining the architecture, modules, interfaces and data.
- Solve a problem or build a product.
- Define the input, output, business rules, data schema.
- Design patterns solve common problems.
- 3 Types:
  - UI design: Data visualization and presentation.
  - Data design: Data representation and storage.
  - Process design: Validation, manipulation and storage of data.

WOMEN WHO CODE

Backend Engineering

Design Patterns

- Set of template solutions that can be reused.
- Improved code maintainability, reusability and scaling.
- Leverages Object-oriented programming (OOP) principles for flexible and maintainable designs.
- Shared pattern vocabulary.
- Not a library or framework, but recommendations for code structuring and problem solving.
- Adapt a pattern and improve upon it to fit application needs.
- Defines relationship between objects, loosely coupled objects, secure code.

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

WOMEN WHO CODE

Backend Engineering

Types of Design Patterns

- **Creational**:
  - Initialize a class and instantiate the objects.
  - Decoupled from implementing system.
  - Singleton, Factory, Builder.
  - Abstract Factory, Prototype.
- **Structural**:
  - Class structure and composition.
  - Increase code reusability and functionality.
  - Create large objects relationships.
  - Adapter, Facade, Decorator, Bridge, Composite, Flyweight, Proxy.
- **Behavioural**:
  - Relationship and communication between different classes.
  - Observer, Strategy, Iterator, etc.

Creational	Structural	Behavioral
• Factory Method	• Adapter	• Interperter
• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Visitor

WOMEN WHO CODE

Backend Engineering

Previously, we discussed...

- Prerequisites for design patterns:
  Basics of programming & OOP
- Necessity of design patterns:
  - Template solutions/ shared vocabulary.
  - Build code on-top of a pattern solution.
  - Maintainability: Easy to maintain code.
  - Reusability: Easy to reuse code for new features.
  - Scaling: Large-scale reuse of architectures.
- Types of design patterns: Creational, Structural, Behavioral
- Solving problems using design patterns and code demos, and real-life applications.
  Examples: Order generation [Bridge/Structural],
  create different shapes [Factory/Creational],
  iterate over a collection [Iterator/Behavioral], etc.

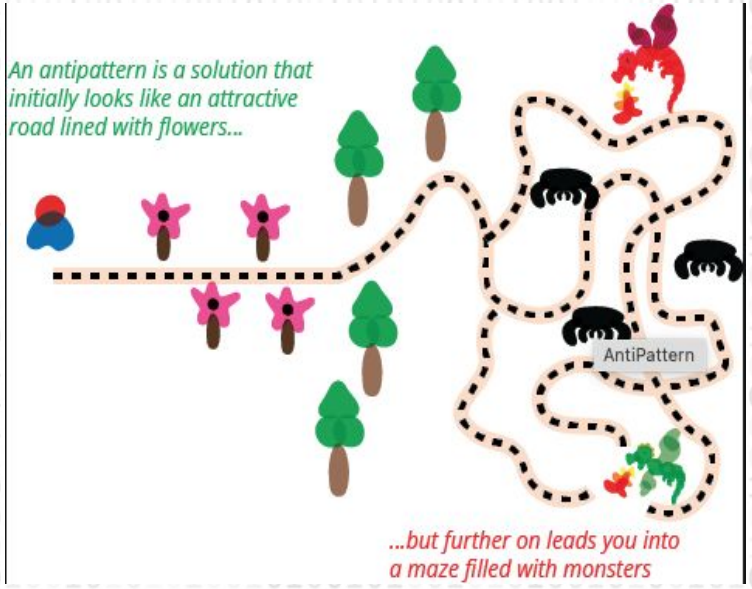
You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

Anti-patterns

- Process or action that doesn't solve a problem and has bad consequences.
- It may seem to solve an existing and/or recurring problem but is ineffective in doing so.
- Identifies inefficient code, bad implementation and coding practices/standards.
- Applies to design, implementation, integration, application behavior, and testing.
- Examples:
  - Big ball of mud: System with no structure.
  - Input Kludge: Does not handle invalid input.
  - Boat anchor: Retaining useless code.



WOMEN WHO CODE

Backend Engineering

Types of Anti-patterns

- Business Operations:
  - Organizational: Deny different viewpoints, bad management, isolated teams, etc.
  - Project management: Underestimate timelines, poor resourcing, etc.

- **Software engineering:**
  - **Software design**: Poorly designed applications that are buggy and unscalable.
  - **Object-oriented programming**: Inefficient code and lacks OOP foundations.
  - **Programming**: Bad programming practices.
  - Methodological: Hard to maintain and upgrade code and improve solutions.
  - Configuration management: Problems with configuration and deployment.

Backend Engineering

- What is an Anti-pattern?

- Can you give examples?

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

• What is an Anti-pattern?
A solution that seems to solve a problem but is ineffective in doing so and may introduce bugs and complexities into the system/software/process.
Bad practices for coding and design.

• Can you give examples?
A monolithic application problems:

- Contained single logical unit with user interface, data access and business logic.
- Large codebase therefore, hard to maintain, debug, and test.
- Tight coupling between modules. And, software is not scalable.
- Useless code still persists and is not removed for features + tests.
- Code from old technologies is running and degrades the performance over time.
- As application evolves, it hard to upgrade code for error handling + integration.
- Everything works or errors break the whole app.

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

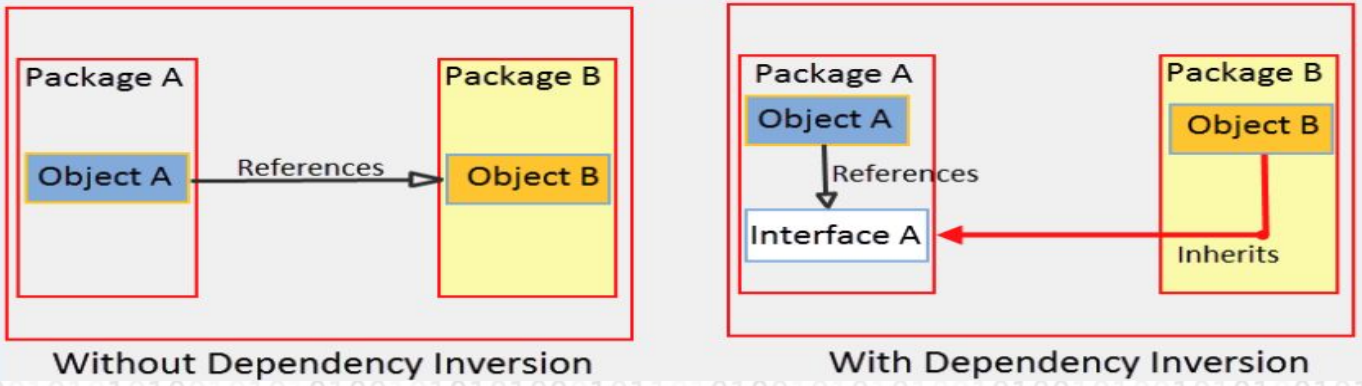
Software design anti-patterns

- How well is the application designed?
- Is it buggy, unscalable and/or manipulates data correctly?
- Poor software design and design patterns not referenced.
- Types:
  - Abstraction inversion
  - Ambiguous viewpoint
  - Big ball of mud
  - Database-as-IPC
  - Inner-platform effect
  - Input kludge
  - Interface bloat
  - Magic pushbutton
  - Race hazard
  - Stovepipe system

Backend Engineering

Software design anti-pattern types

- **Abstraction inversion**: Functionality is not exposed by the code (interface) therefore, caller method re-implements the functionality.
  - Duplicated code, hard to match updates, more errors, inefficient and costly.
  - Examples: Running many queries instead of stored procedures (if no access), use interface and abstraction to access operations.



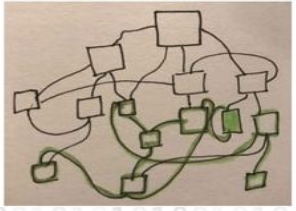
WOMEN WHO CODE

Backend Engineering

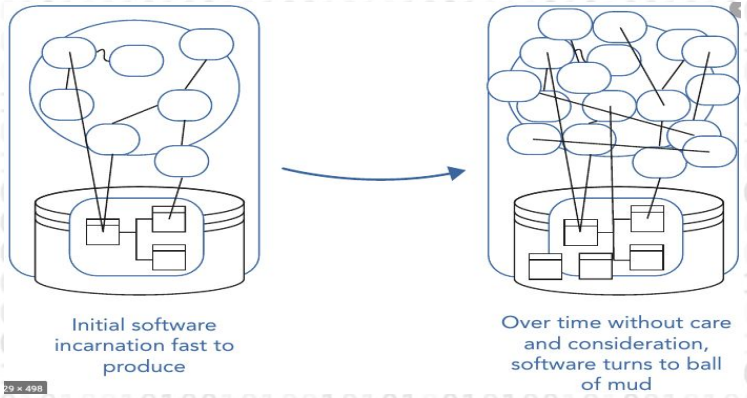
Software design anti-pattern types

- **Big Ball of Mud**: Application lacks architecture and isn't cohesive.
  - Code is old/obsolete, not suitable for optimization, highly buggy, etc.
  - AKA Spaghetti code (unstructured code) or technical debt (need to rewrite the code).
  - Examples: Small set of services dependent on each other. Over time, more dependencies, more path flows, and tight coupling.

Big Ball of Mud Architecture



"You reach for the banana, and get the entire gorilla"
– Michael Stahl



WOMEN WHO CODE

Backend Engineering

Software design anti-pattern types

- **Input Kludge (collection)**: Application fails to handle erroneous/ invalid input.
  - Incorrect input: Bad request (fails validations), incorrect data format (json/xml).
  - Users can easily crash the system as errors are unhandled so the app errors-out.
  - Extensive testing for bad input needs to be done.
  - Example: REST API gets bad/incorrect data input.

"dateOfBirth" and "advisor" fields are required.

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "dateOfBirth": "1990-01-12",
    "yearOfEnrollment": "2021",
    "degree": "COMPUTER_SCIENCE",
    "advisor": "Prof. Matt Peterson"
}
```

```
{
  "firstName": "Jane",
  "lastName": "Doe",
  "yearOfEnrollment": "2021",
  "degree": "COMPUTER_SCIENCE"
}
```

Backend Engineering

- What is a Software design anti-pattern?
- What are the types of software design anti-patterns?
- Can you give examples?

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

- What is a Software design anti-pattern?

Bad practices to design and structure application code.

- What are the types of software design anti-patterns?

Big ball of mud, abstraction inversion, input kludge.

- Can you give examples?

Duplicated code, bad data input, lots of bugs, obsolete coding practices followed.

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

Object-oriented anti-patterns

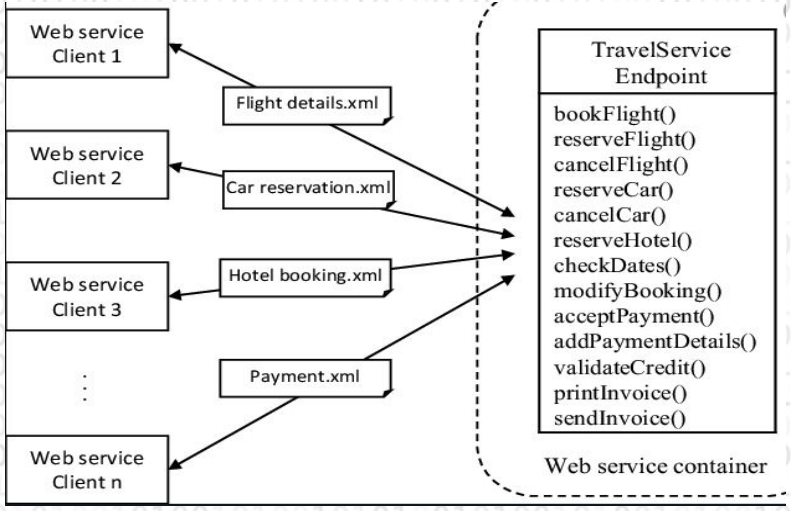
- Poorly implemented object-oriented programming principles.
- Polymorphism, inheritance, abstraction, encapsulation, interface & object relationships.
- Types:
  - Anemic domain model
  - Call super
  - Circle–ellipse problem
  - Circular dependency
  - Constant interface
  - God object
  - Object cesspool
  - Object orgy
  - Poltergeists
  - Sequential coupling
  - Singleton Pattern
  - Yo-yo problem

WOMEN WHO CODE

Backend Engineering

Object-oriented anti-pattern types

- **God Object**: An entity/object has many functions that complicate implementation.
  - Inefficient bifurcation of a large problem into smaller problems.
  - Tight coupling with an object for all functionalities and data.
  - Object exclusively stores state management.
  - Single point of failure.
  - Example: For cars, flights, hotels.



WOMEN WHO CODE

Backend Engineering

Object-oriented anti-pattern types

- **Object Orgy**:
  - Objects are incorrectly encapsulated, causing security concerns.
  - Unrestricted access to object state.
  - Enforces program security via Encapsulation.
  - Example: Variables not marked as "private" instead marked "public" and getter() and setter() methods cause faulty data and behavior changes/ abuse.

```
# program to illustrate public access modifier in a class

class Geek:

    # constructor
    def __init__(self, name, age):

        # public data mambers
        self.geekName = name
        self.geekAge = age

    # public memeber function
    def displayAge(self):

        # accessing public data member
        print("Age: ", self.geekAge)

# creating object of the class
obj = Geek("R2J", 20)

# accessing public data member
print("Name: ", obj.geekName)

# calling public member function of the class
obj.displayAge()
```


WOMEN WHO CODE

Backend Engineering

- What is an Object-oriented anti-pattern?
- What are the types of object-oriented anti-patterns?
- Can you give examples?

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

• What is an Object-oriented anti-pattern?
No/incorrect implementation of OOP principles.

• What are the types of object-oriented anti-patterns?
God object, object orgy, circular dependency.

• Can you give examples?
Unsafe access to objects, object loaded with functionality, objects circularly dependent on each other, interface defines constants (is not supposed to store state).

You can unmute and talk or use the chat.

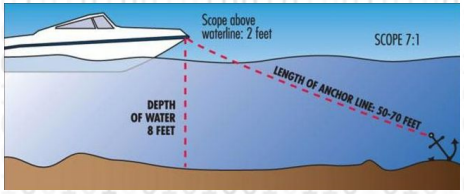
WOMEN WHO CODE

Backend Engineering

Programming anti-patterns

- Bad programming practices.
- Over-engineering problems, using the wrong toolset, grouping unrelated issues/features which can cause complexities, etc.
- Types:
  - Accidental complexity
  - Action at a distance
  - Boat anchor
  - Busy waiting
  - Caching failure
  - Cargo cult programming
  - Coding by exception
  - Design pattern
  - Error hiding
  - Hard code
  - Lasagna code
  - Lava flow
  - Loop-switch sequence
  - Magic numbers
  - Magic strings
  - Repeating yourself
  - Shooting the messenger
  - Shotgun surgery
  - Soft code
  - Spaghetti code

Backend Engineering



Programming anti-pattern types

- **Boat Anchor**: Throw-away or obsolete code is retained.
  - Difficulty differentiating between working and obsolete code.
  - Either delete the code or mark it as deprecated, or move/isolate the code.
  - Examples: Poor/no documentation, convoluted implementation, C# *Obsolete* attribute, Python *@deprecated* decorator.
  - A metaphor to throwing an anchor in the water.

```python
import warnings

from deprecated.sphinx import deprecated

warnings.filterwarnings("always")


@deprecated(version='1.0', reason="This function will be removed soon")
def some_old_function(x, y):
    warnings.warn("some_old_function is deprecated", DeprecationWarning)
    return x + y


if __name__ == '__main__':
    some_old_function(12, 34)
    help(some_old_function)
```

```csharp
[Obsolete]
public void ObsoleteMethod() { }

[Obsolete("This method is deprecated, is better to use another method")]
public void ObsoleteMethodWithMessage() { }

[Obsolete("This method is deprecated, and cause to not failed compilation", false)]
public void ObsoleteMethodWithMessageAndNotFailedCompiling() { }

[Obsolete("This method is deprecated, and cause to failed compilation", true)]
public void ObsoleteMethodWithMessageAndFailedCompiling() { }
```


WOMEN WHO CODE

Backend Engineering

Programming anti-pattern types

- **Hard Coding**: Embedding data into the program instead of fetching at runtime.
  - Any change in values requires source code changes, recompilation and retesting.
  - End-user or downstream system needs to be made aware of the changes.
  - Backdoor: Security concern if hard-coded credentials.
  - Magic number: If hard-coded value is repeated then it is hard to update instances.
    - Example:

Before:

```
double areaOfCircle(double radius) {
  return radius * radius * 3.1415;
}
```

After:

```
static final double PI_VALUE = 3.1415;
double areaOfCircle(double radius) {
      return radius * radius * PI_VALUE;
}
```


WOMEN WHO CODE

Backend Engineering

Programming anti-pattern types

- Example: Android layout has hard-coded margins, font sizes, etc.
  So, redefined them into sizes.xml file and referred the file in the layout.

```
<EditText
    android:id="@+id/post_title"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:textSize="22sp"
    android:hint="Title"
    android:inputType="textCapSentences|textAutoCorrect" />
```

```
<dimen name="margin_small">4dp</dimen>
<dimen name="margin_medium">8dp</dimen>
<dimen name="margin_large">12dp</dimen>

<dimen name="text_sz_small">12sp</dimen>
<dimen name="text_sz_medium">14sp</dimen>
<dimen name="text_sz_large">18sp</dimen>
<dimen name="text_sz_extra_large">22sp</dimen>
```

```
<EditText
    android:id="@+id/post_title"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/margin_medium"
    android:layout_marginRight="@dimen/margin_medium"
    android:layout_marginTop="@dimen/margin_medium"
    android:textSize="@dimen/text_sz_extra_large"
    android:hint="@string/title"
    android:inputType="textCapSentences|textAutoCorrect" />
```


WOMEN WHO CODE

Backend Engineering

- What is a Programming anti-pattern?
- What are the types of programming anti-patterns?
- Can you give examples?

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

- What is a Programming anti-pattern?

Incorrect/No use of programming principles.

- What are the types of programming anti-patterns?

Hard coding, boat anchor, unexplained magic numbers, spaghetti code, storing business logic in configuration files rather than at the service level in the code, etc.

- Can you give examples?

Magic numbers, hard-coded strings, DRY don't repeat yourself (no code duplications), open to using different or better toolsets, etc.

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Engineering

Summary:

- Anti-patterns
- Problems: Code that is not maintainable, hard to debug and fix, does not implement business logic correctly, hard to upgrade/modify, etc.
- Software design anti-pattern and types.
- Object-oriented programming anti-pattern and types.
- Programming anti-pattern and types.
- Benefits: Identify faulty code, improve unsafe code, refactor/rewrite code, update code with best practices, increase efficiency, performance and resource management, agility to implement new features and less bugs, easy to debug and log.

NEXT SESSION on 7-1-2021: Design patterns' interview questions and coding.
Come prepared with questions!

You can unmute and talk or use the chat.

WOMEN WHO CODE

Backend Study Group

- WWCode Presentation and Demo
- WWCode YouTube channel:
  - March 25, 2021 session recording: Backend Engineering
  - April 8, 2021 session recording: Java Microservice and REST API Demo
  - April 22, 2021 session recording: Creational Design Patterns
  - May 20, 2021 session recording: Structural Design Patterns
  - June 3, 2021 session recording: **Behavioral Design Patterns**
- **Resources**:
  - Software design pattern
  - Design Patterns in Java
  - Design Patterns in Python and Ruby
  - Head First Design Patterns book
  - Anti-pattern
  - GeeksForGeeks
  - Android Tip: Avoid Hard-coded Sizes

You can unmute and talk or use the chat.

WOMEN WHO CODE