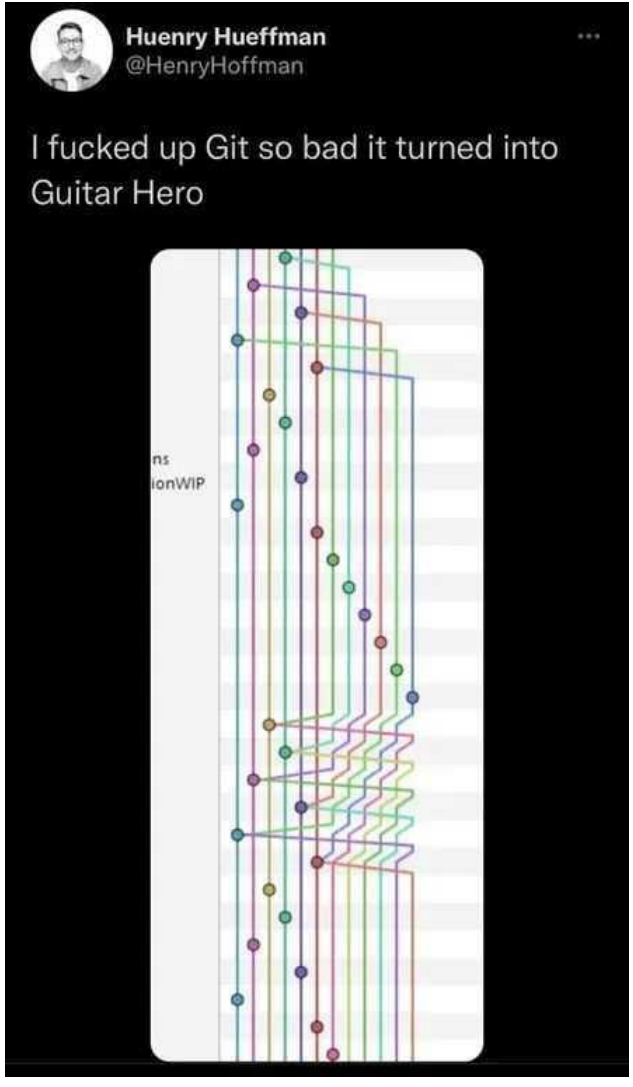
AARHUS UNIVERSITET

Software Engineering and Architecture

Comments on Mandatory


Huenry Hueffman
@HenryHoffman
I fucked up Git so bad it turned into Guitar Hero
ns
ionWIP

AARHUS UNIVERSITET

State Pattern

- The point about ZetaStone was *reuse* ☺
  - Correct state pattern

```
public class ZetaWinningStrategy implements WinningStrategy {
    private GameState currentState;  5 usages
```

```
public Player getWinner(Game game) {
    int roundNumber = game.getTurnNumber();

    if (roundNumber > 12) {
        currentState = new LateGameState();
    } else if (roundNumber > 6) {
        currentState = new MidGameState();
    } else {
        currentState = new EarlyGameState();
    }

    return currentState.getWinner(game);

}
```

  - … but
    - Each of the State's are **source-code-copy**
    - Not **reusing** the WinnerStrategy but **recoding it**

```
public interface GameState {  9 u
    Player getWinner(Game game);
}
```

Abstract Factory

- *Family of objects*

[13.1] Design Pattern: Abstract Factory

Intent	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Problem	Families of related objects need to be instantiated. Product variants need to be consistently configured.
Solution	Define an abstraction whose responsibility it is to create families of objects. The client delegates object creation to instances of this abstraction.

- *Consider:* Game needs a Deck of cards
  - One feasible Strategy is to 'List<Card> createDeck(…)'

Abstract Factory

- *Family of objects*

[13.1] Design Pattern: Abstract Factory

Intent	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Problem	Families of related objects need to be instantiated. Product variants need to be consistently configured.
Solution	Define an abstraction whose responsibility it is to create families of objects. The client delegates object creation to instances of this abstraction.

- *Alpha Factory creates DeckBuildStrategy creates Deck*
- *Alpha Factory creates Deck*
  - Avoiding the Strategy?

AARHUS UNIVERSITET

Compositional Design

- So compositional design thinking often leads to Design Patterns
  - And sometimes it is another one that you think

- If you have HeroBuildingStrategy and DeckBuildingStrategy
  - (As I have)
- Then they are more like a ‘Abs Factory’ than ‘Strategy’
  - And can probably be removed and replaced by your factory
  - (I have been lazy and have not done so.)

AARHUS UNIVERSITET

Test Stub

- The ‘pick random minion’ is a responsibility which must be encapsulated in a **role** via an **interface** and can be played by two objects
  - The real random object — production code
  - The stubbed object — test code

- How to configure EpsilonStone to use the proper object?

- Analysis
  - Creating an object to generate random indices is a part of the HotStone configuration – and should be in the Abs Factory

AARHUS UNIVERSITET

Configuring Epsilon

- Tedious to create a new Java file with just that one minor change compared to normal Epsilon
  - Class TestEpsilonFactory extends EpsilonFactory { … }

- You can use ‘anonymous classes’ instead…

AARHUS UNIVERSITET

Ala…

- ... In place overwriting a method

```
public class TestEpsilonStone {
    private Game game; 14 usages
    private FixedIndexStrategy indexStrategy; 4 usages

    /** Fixture for AlphaStone testing. */
    @BeforeEach
    public void setUp() {
        indexStrategy = new FixedIndexStrategy();
        class TestEpsilonStoneFactory extends EpsilonStoneFactory {
            @Override 1 usage
            public HeroStrategy setUpHeroStrategy() { return new FrenchAndItalianHeroStrategy(indexStrategy); }
        }

        game = new StandardHotStoneGame(new TestEpsilonStoneFactory());
    }
```

- Critique: Methods should be called ‘create…’ or similar