HUSTEF

HUNGARIAN SOFTWARE TESTING FORUM

Design patterns to boost your test automation

Christian Baumann

@chrissbaumann @chrissbaumann@sw-development-is.social

Who is that guy?

Christian Baumann

Software Tester

MAIBORNWOLFF

@chrissbaumann @chrissbaumann@sw-development-is.social

How is your test automation journey going?

Design Patterns: Definition

... a description of customized communicating objects and classes that solves a problem in a particular context of software design.

Gang of Four

Design Patterns: Definition

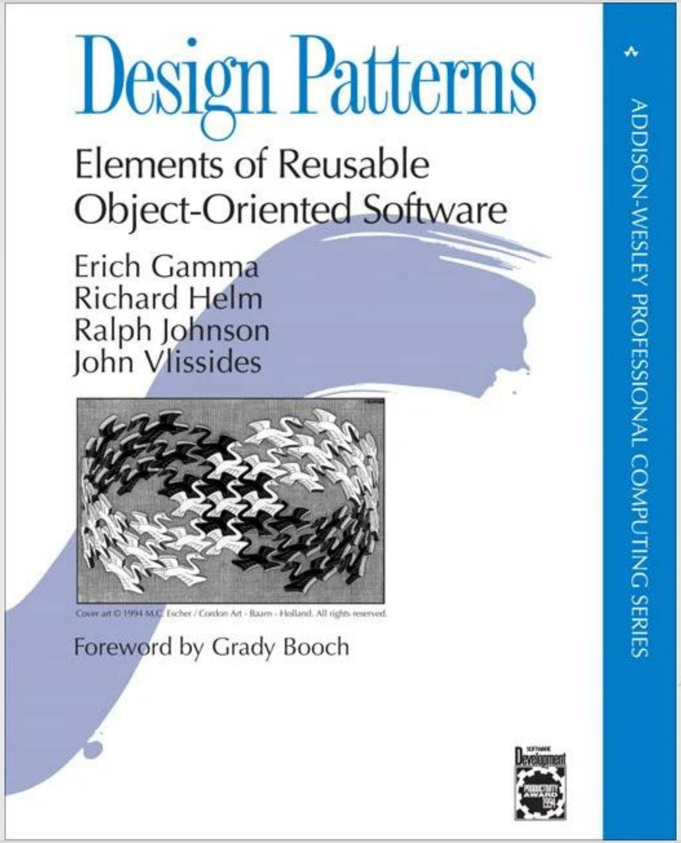
A design pattern is a common way of building things that solves a known problem.

Content of this talk

- Some history & theory
- Design patterns for test automation
- How to get started
- Drawbacks & limitations

How it all started

The book


Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Types of Design Patterns

- Creational
- Behavioral
- Structural

Types of Design Patterns: Creational

- creating objects while hiding the creation logic
- gives more flexibility in deciding for objects depending on the use case

Types of Design Patterns: Behavioral

- deal with class and object composition
- Inheritance to
  - compose interfaces
  - add new functionalities to objects

Types of Design Patterns: Structural

- communication between objects

Elements of a design pattern

- Name
- Problem
- Solution
- Consequences

Design pattern: Name

- one or two word description
- used by programmers familiar with the pattern
- should recall problem and solution

Design pattern: Problem

- general intent
- one or two specific motivations

Design pattern: Solution

- specifies elements that make up the pattern
- includes relationships, responsibilities and collaborators

Design pattern: Consequences

- often more than one pattern matching a problem
- consequences of patterns become a determining factor

What design patterns are not!

- not algorithms
- you cannot select and paste it
- not a specific piece of code

SOLID principles

SOLID principles

- Single Responsibility
- Open/ Closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

Examples of Design Patterns in Test Automation

- Builder Pattern
- Decorator Pattern
- Strategy Pattern
- Singleton Pattern
- Page Object Pattern
- Screenplay Pattern

Builder Pattern

Builder Pattern: Usages

- algorithm of complex object creation should be independent
- construction process must allow different representations
- tests are often bound to a constructor
- builder pattern can resolve this dependency

Builder Pattern: Production vs. Test

- Production: encapsulation, constraints & limitations

- Test: expose units & test in isolation

Builder Pattern: A typical domain class

```
public class Employee {
    public mployee(int id, string firstname,
                    string lastname, DateTime birthdate, string street) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthdate = birthdate;
        this.street = street;
    }

    public String getFullName() {
        return this.firstName + " " + this.lastName;
    }

    public int getAge() {
        int age = LocalDate.now.getYear() - dateOfBirth.getYear();
        LocalDate other = today.plusYears(-age);
        if (dateOfBirth.isAfter(other)) { age--; }
        return age;
    }
}
```

Builder Pattern: A simple unit test

```
@Test
public void getFullNameReturnsCombination() {
    Employee employee = new Employee(   1,
                                        "Chriss",
                                        "Baumann",
                                        new DateTime(1980, 1, 8),
                                        "Homestreet");


    fullname = employee.getFullName();

    assertThat(fullname, Is.EqualTo("Chriss Baumann"));
}
```

Builder Pattern: A simple unit test - problems & consequences

- Problem: tied to the constructor
- Irrelevant data needs to be given

- Consequences:
  - Test lacks expressiveness
  - Test breaks when adding item to constructor

Builder Pattern: A simple unit test - different goals

Problem with two different goals:

- Production code should be immutable
- For tests we only want to provide some data

Builder Pattern: A class with a method that creates an object

```
public class EmployeeBuilder {
    private int id = 1;
    private string firstname = "first";
    private string lastname = "last";
    private DateTime birthdate = DateTime.Today;
    private string street = "street";

    public Employee Build() {
        return new Employee(id, firstname, lastname, birthdate, street);
    }
}
```

Builder Pattern: Modification methods

```
public void withFirstName(string firstname) {
    this.firstname = firstname;
}

public void withLastName(string lastname) {
    this.lastname = lastname;
}

EmployeeBuilder builder = new EmployeeBuilder();
builder.withFirstName("Chriss");
builder.withLastName("Baumann");
Employee employer = builder.Build();
```

Builder Pattern: Fluent API

```
public EmployeeBuilder withFirstName(string firstname) {
    this.firstname = firstname;
    return this;
}

public EmployeeBuilder withLastName(string lastname) {
    this.lastname = lastname;
    return this;
}

EmployeeBuilder builder = new EmployeeBuilder().
        withFirstName("Chriss").
        withLastName("Baumann");
Employee employee = builder.Build();
```

Builder Pattern: Advantages

- hides irrelevant details
- expressive
- flexible
- reliable

Decorator Pattern

Decorator Pattern

```
public interface Coffee {

    public double getCost();

    public String getIngredients();
}

public class SimpleCoffee implements Coffee {
    @Override
    public double getCost() {
        return 1;
    }

    @Override
    public String getIngredients() {
        return "Coffee";
    }
}
```

Decorator Pattern

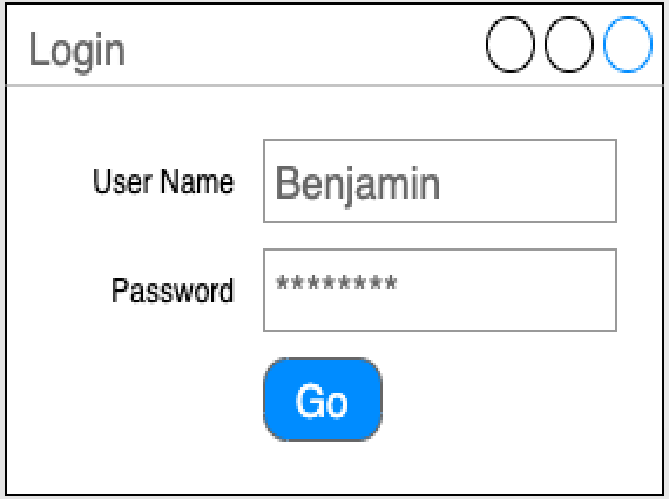
```java
public abstract class CoffeeDecorator implements Coffee {
    private final Coffee decoratedCoffee;

    public CoffeeDecorator(Coffee c) {
        this.decoratedCoffee = c;
    }

    @Override
    public double getCost() {
        return decoratedCoffee.getCost();
    }

    @Override
    public String getIngredients() {
        return decoratedCoffee.getIngredients();
    }
}
```

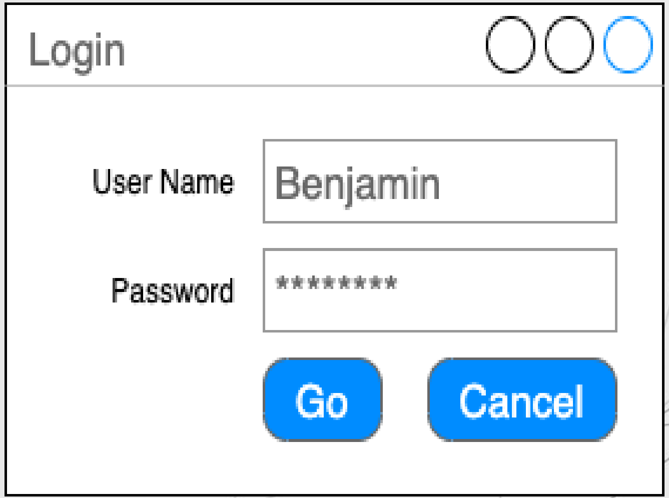
```java
class WithMilk extends CoffeeDecorator {
    public WithMilk(Coffee coffee) {
        super(coffee);
    }

    @Override
    public double getCost() {
        return super.getCost() + 0.5;
    }

    @Override
    public String getIngredients() {
        return super.getIngredients() + ", Milk";
    }
}

class WithSprinkles extends CoffeeDecorator {
    public WithSprinkles(Coffee coffee) {
        super(coffee);
    }

    @Override
    public double getCost() {
        return super.getCost() + 0.2;
    }

    @Override
    public String getIngredients() {
        return super.getIngredients() + ", Sprinkles";
    }
}
```

Decorator Pattern

```
public class Main {
    public static void printInfo(Coffee c) {
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());
    }

    public static void main(String[] args) {
        Coffee coffee = new SimpleCoffee();
        printInfo(coffee);  // Cost: 1.0; Ingredients: Coffee

        Coffee coffeeWithMilk = new WithMilk(coffee);
        printInfo(coffeeWithMilk);  // Cost: 1.5; Ingredients: Coffee, Milk

        Coffee coffeeWithSprinkles = new WithSprinkles(coffee);
        printInfo(coffeeWithSprinkles); //Cost: 1.2; Ingredients: Coffee, Sprinkles

        Coffee coffeeWithMilkAndSprinkles = new WithSprinkles(coffeeWithMilk);
        printInfo(coffeeWithMilkAndSprinkles);  // Cost: 1.7; Ingredients: Coffee, Milk, Sprinkles
    }
}
```

Decorator Pattern: Benjamin Bischoff's example





Decorator Pattern: Usage

- add responsibilities dynamically and transparently
- for responsibilities that can be withdrawn
- when extension by subclassing is impractical

Strategy Pattern

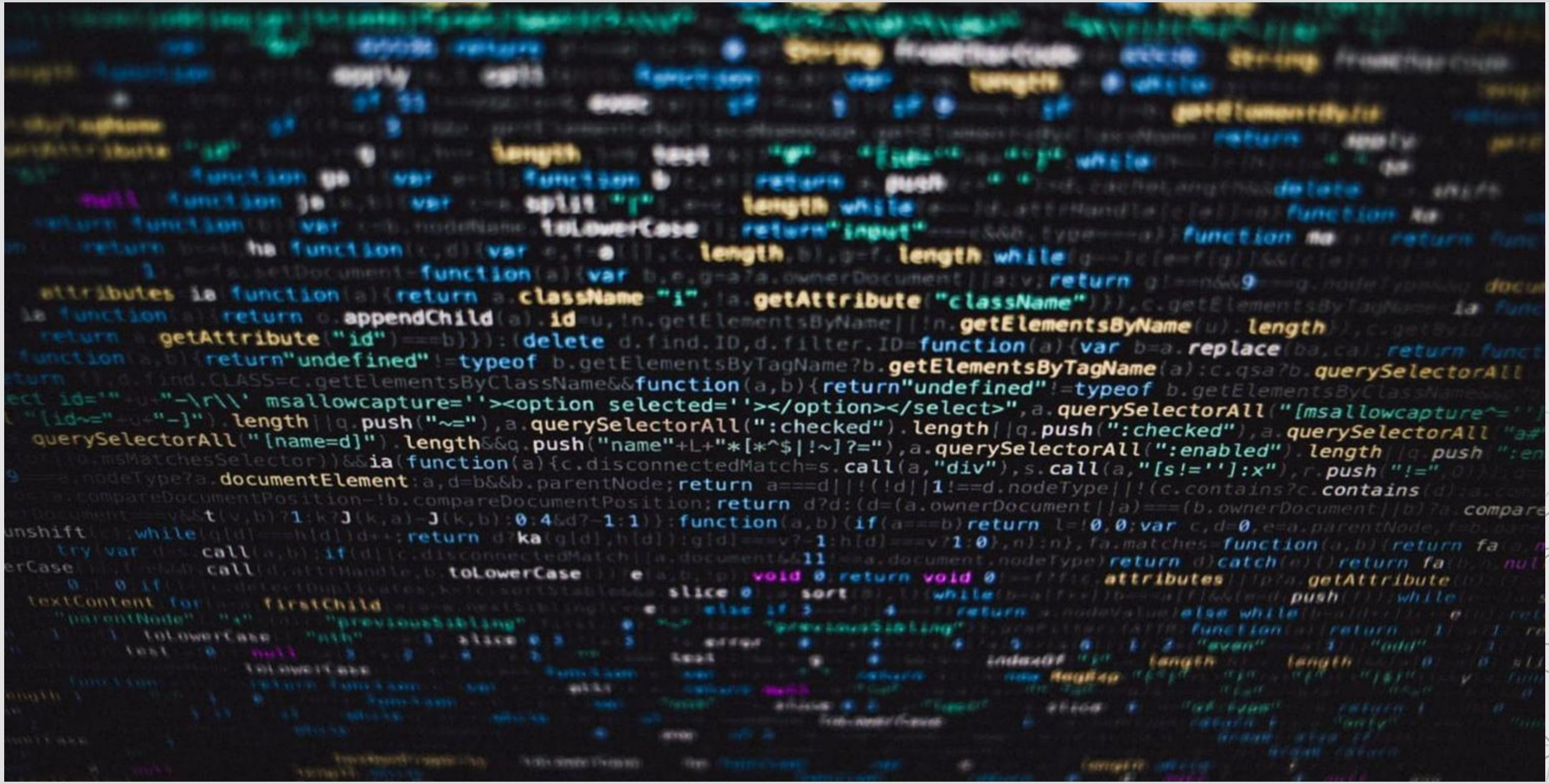
Strategy Pattern

```
public interface UserRegistrationStrategy {
     User register();
 }

class WebUserRegistrationStrategy implements UserRegistrationStrategy {
    @Override
    public User register() {
        String username = UserRegistry.getUsername();
        String password = PasswordGenerator.generatePassword();
        SignInPage.open()
                .pressSignUpButton()
                .enterUsername(username)
                .enterPassword(password)
                .clickRegisterButton();
        return new User(username, password);
    }
}

class ApiUserRegistrationStrategy implements UserRegistrationStrategy {
    @Override
    public User register() {
        User user = new User(UserRegistry.getNewUsername(), PasswordGenerator.
        generatePassword());
        put("api/user").withBody(user.toJson());
        return user;
    }
}
```

Strategy Pattern: Usage

- many related classes differ only in behavior
- different variants of an algorithm
- algorithm uses data that clients shouldn't know
- class defines many behaviours which appear as multiple conditional statements

Singleton Pattern



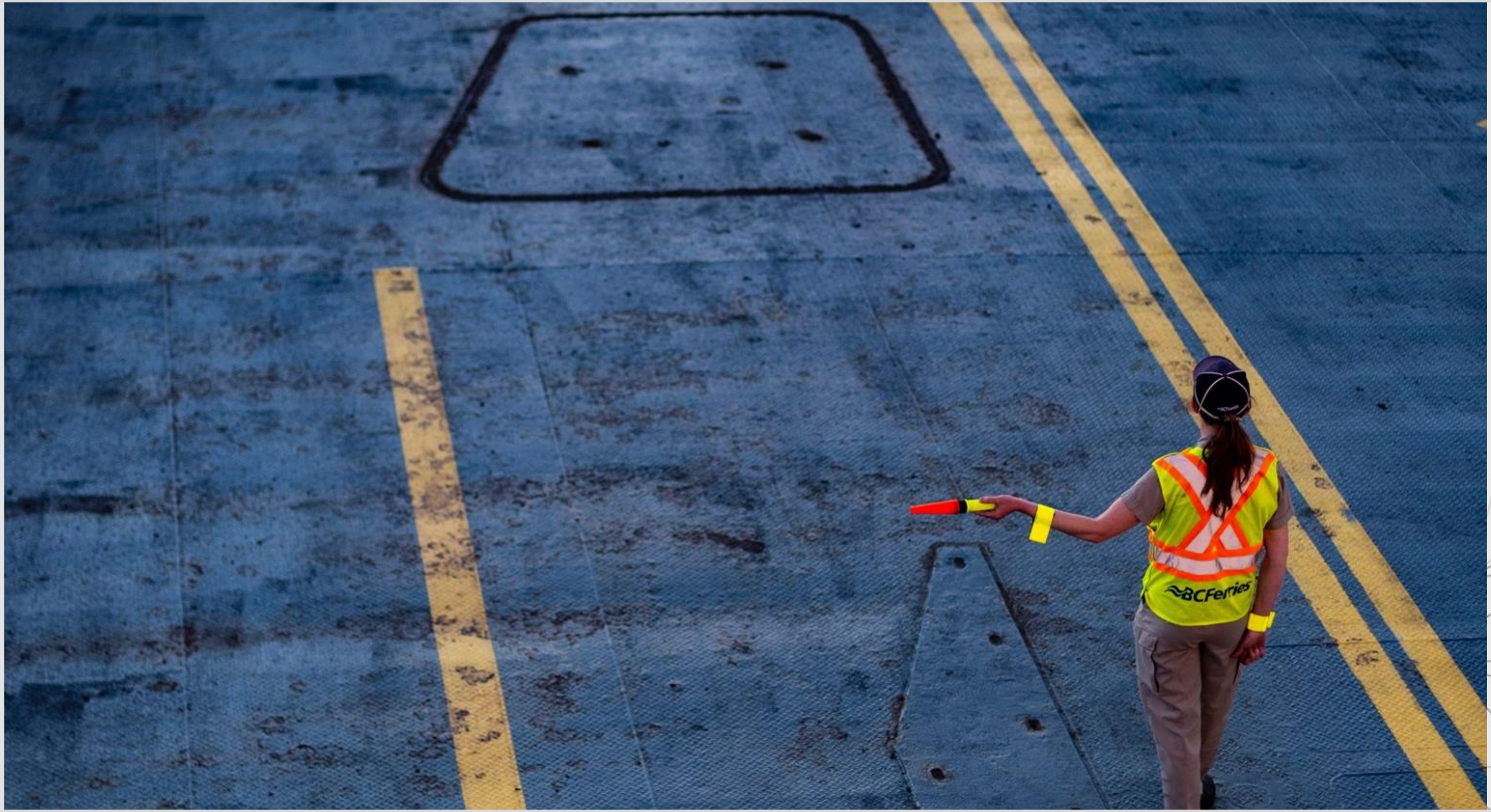
Singleton Pattern

```java
public class SingletonWebDriver {
    public static SingletonWebDriver singletonWebDriver;
    private WebDriver webDriver;

    private SingletonWebDriver() {
        webDriver = new ChromeDriver();
    }

    public static SingletonWebDriver getSingletonWebDriver() {
        if (singletonWebDriver == null)
            singletonWebDriver = new SingletonWebDriver();
        return singletonWebDriver;
    }

    public WebDriver getWebDriver() {
        return webDriver;
    }
}
```

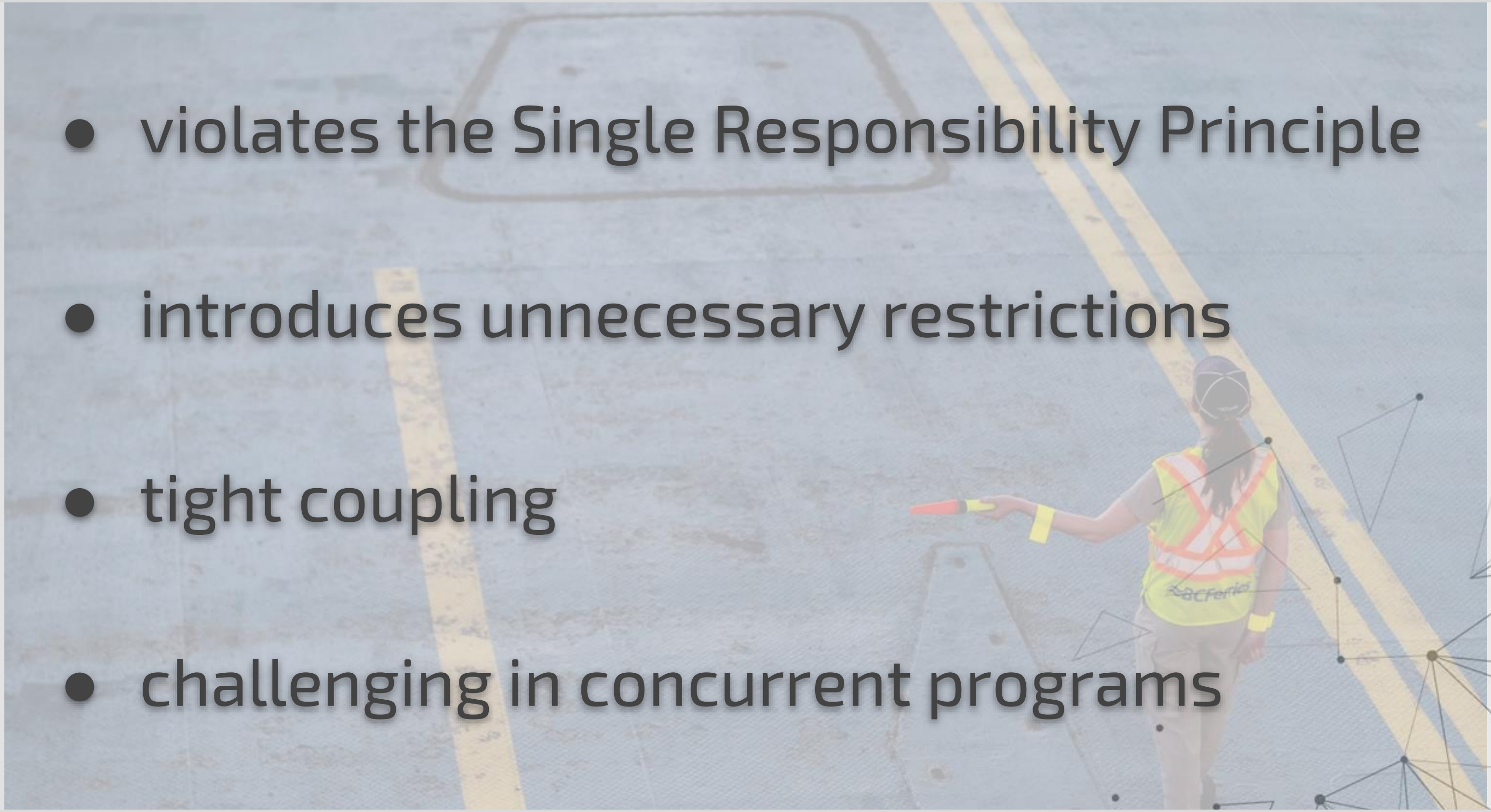
Singleton Pattern: Usage

```
SingletonWebDriver singletonWebDriver =
        SingletonWebDriver.getSingletonWebDriver();

WebDriver webDriver =
        singletonWebDriver.getWebDriver();


SingletonWebDriver anotherWebDriver =
        SingletonWebDriver.getSingletonWebDriver();

WebDriver webDriver =
        another.getWebDriver();
```

Singleton Pattern - an anti pattern?

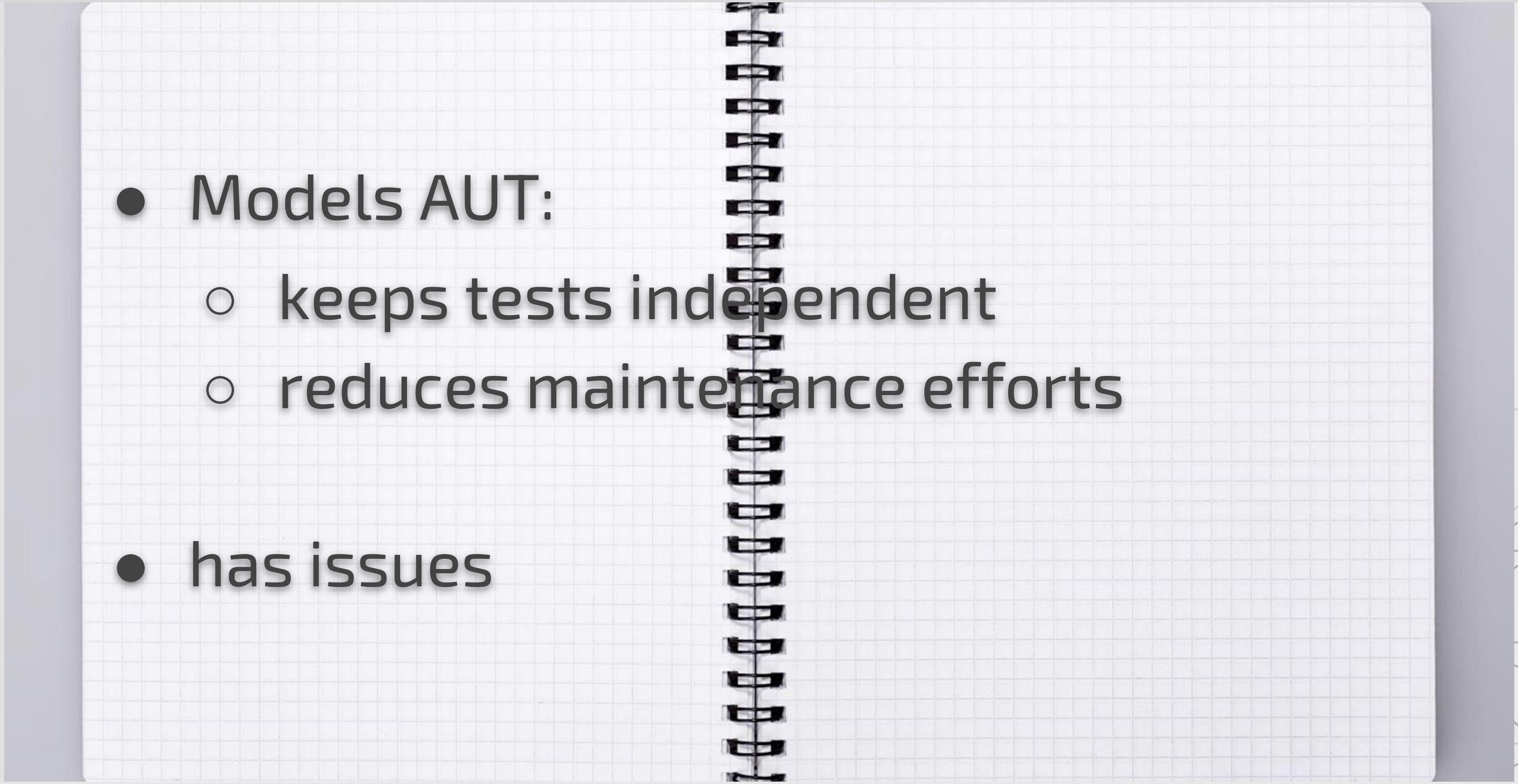


Singleton Pattern - an anti pattern?



- violates the Single Responsibility Principle
- introduces unnecessary restrictions
- tight coupling
- challenging in concurrent programs

Page Object Model



- Models AUT:
  - keeps tests independent
  - reduces maintenance efforts

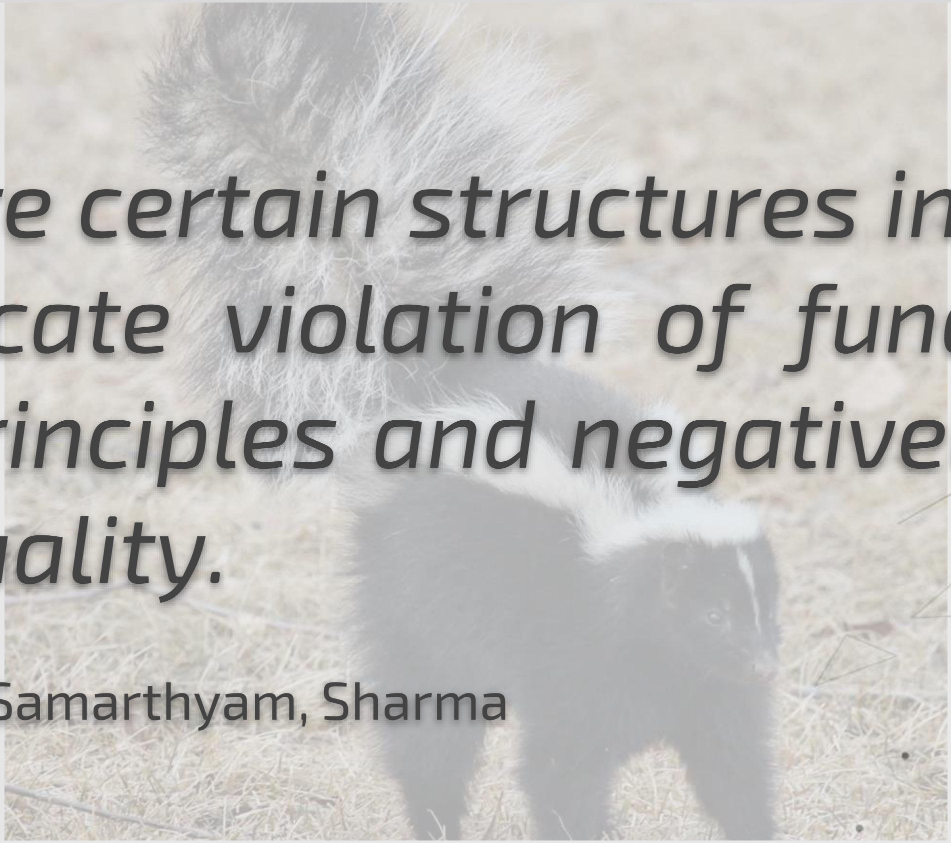
- has issues

Page Object Model

- reduces maintenance issues, wrongly attributed to Selenium, instead of coding practices
- can be a good starting point, but violates good coding practices
- problems solved: repetition & inconsistencies
- “good enough” solution
- causes maintenance overhead

Code smells

Code smells

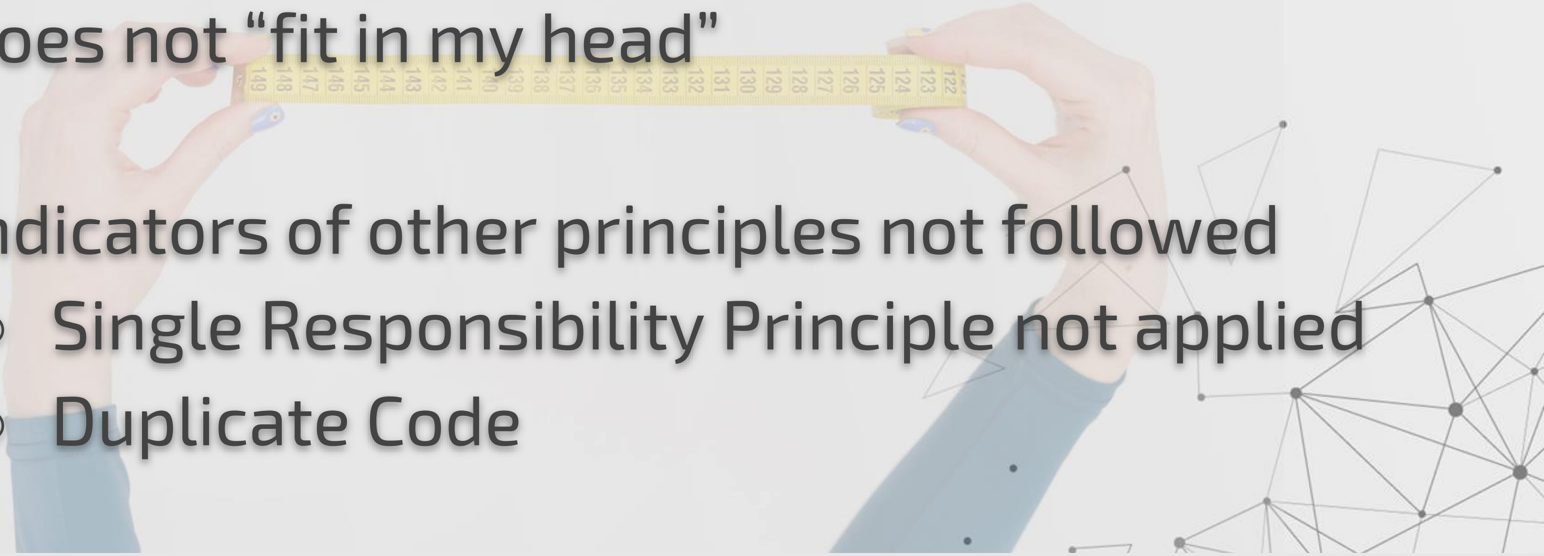


Smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality.

Suryanarayana, Samarthyam, Sharma

Page Object: Long Class

Page Object: Long Class



- hard to find things
- difficult to maintain
- does not “fit in my head”

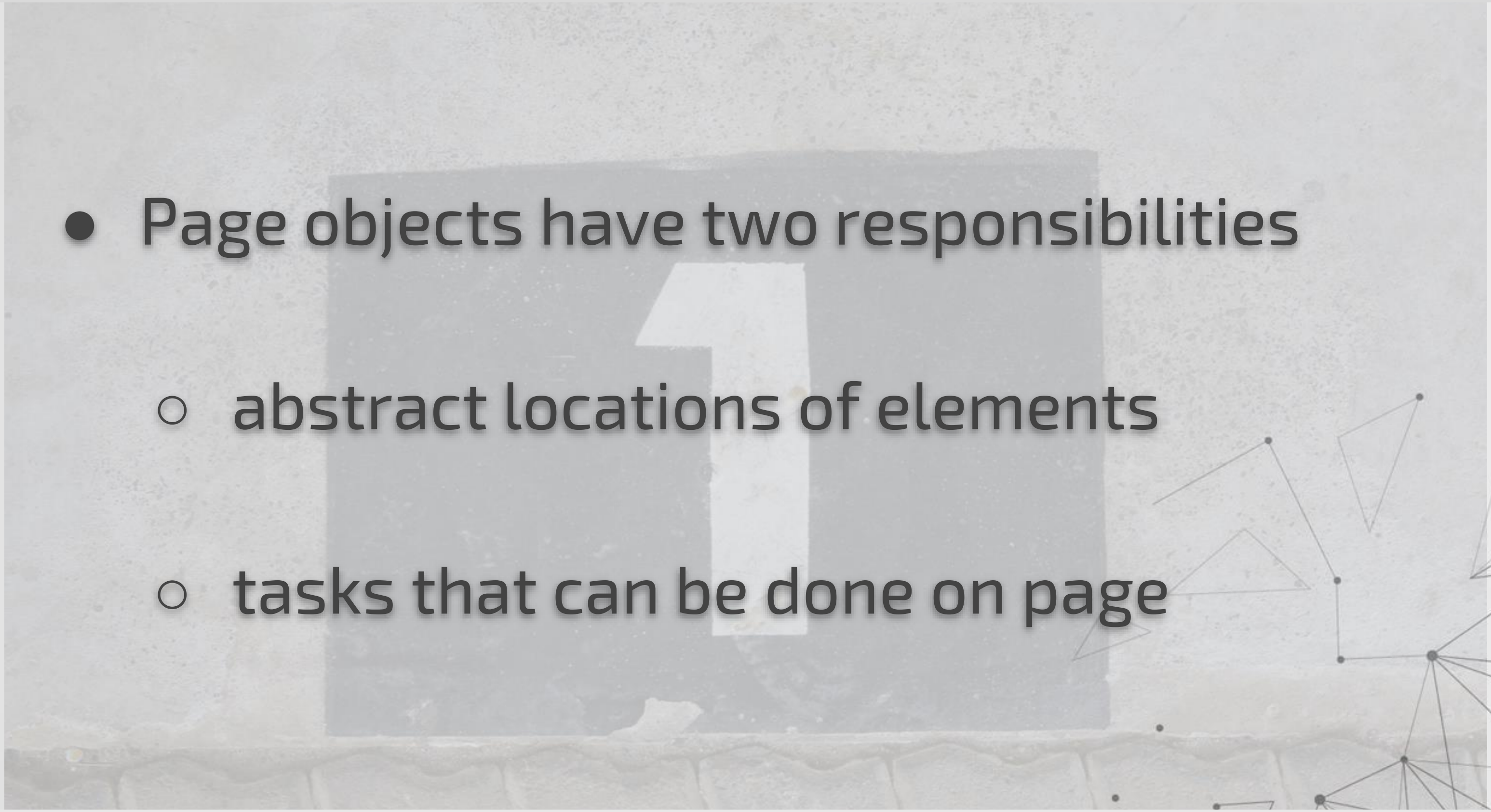
- indicators of other principles not followed
  - Single Responsibility Principle not applied
  - Duplicate Code

Page Object: Some SOLID principles

- **S**ingle Responsibility
- **O**pen/ Closed

Page Object: Single Responsibility

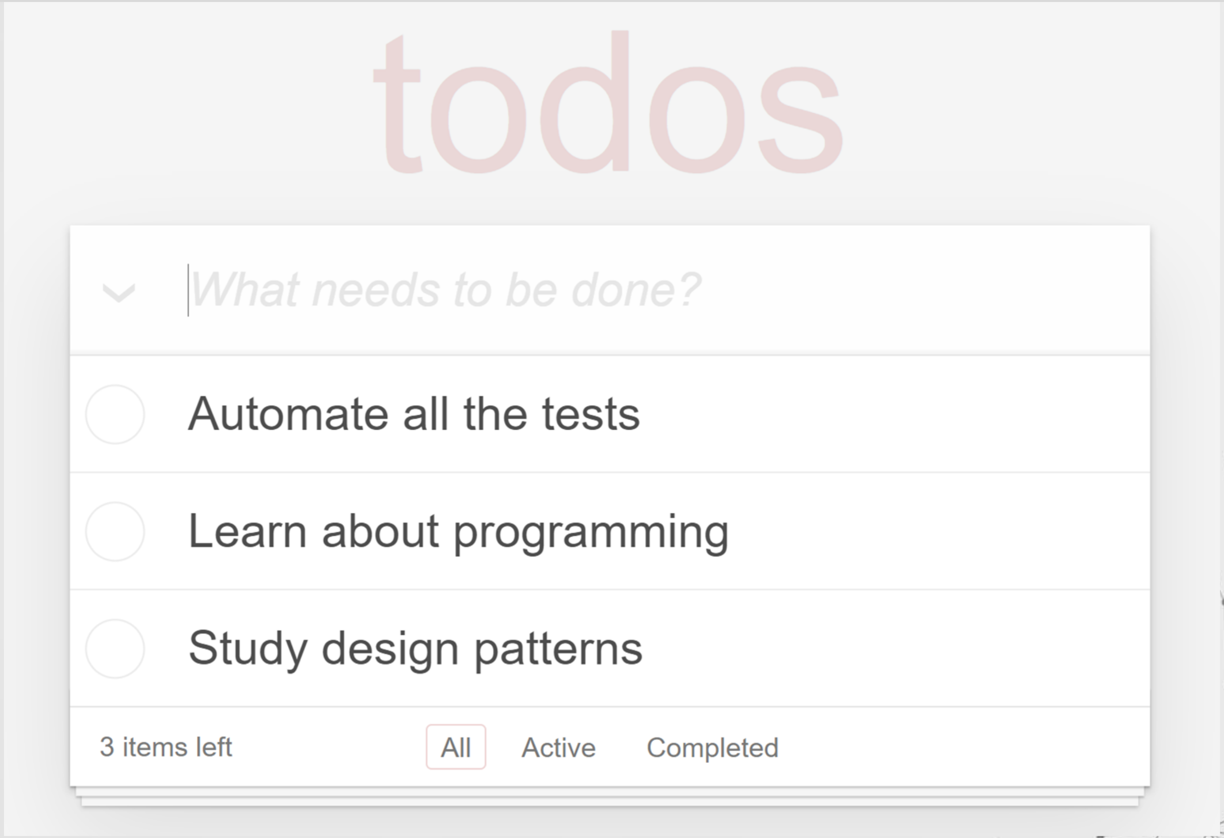
Page Object: Single Responsibility



- Page objects have two responsibilities
  - abstract locations of elements
  - tasks that can be done on page

Page Object: Open Closed

Page Object: Open Closed



Page Object: Open Closed

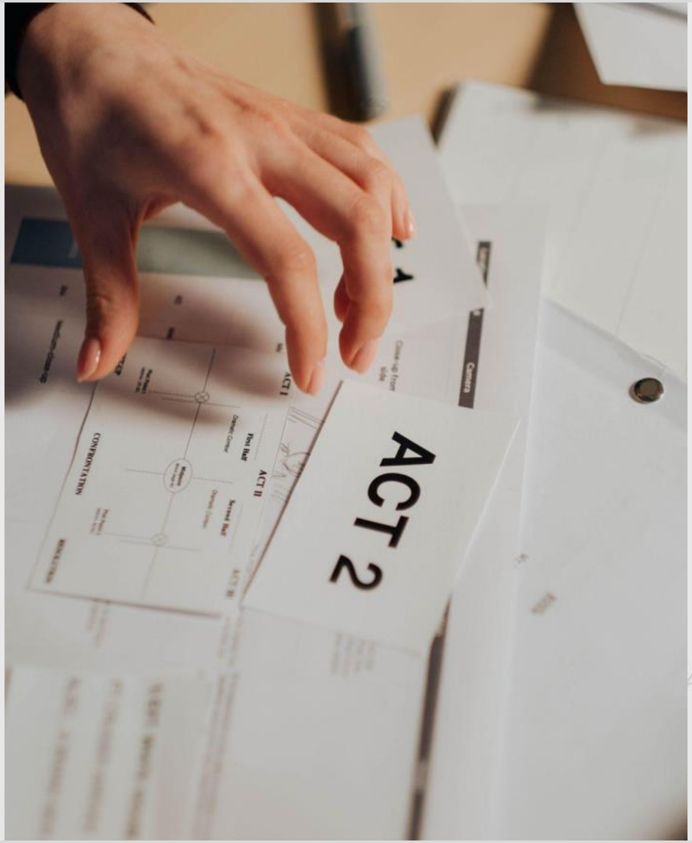
- Satisfy Open Closed principle:
  - split classes
  - new behaviour → new class
  - ⚡ violates Single Responsibility principle

- Satisfy Single Responsibility principle:
  - one class for locating elements & performing tasks each
  - ⚡ violates Open Closed principle

Page Object Model: more issues

- user behaviour spans more than a single page
- “behaviours” to describe user’s actions, then thinking in “pages”

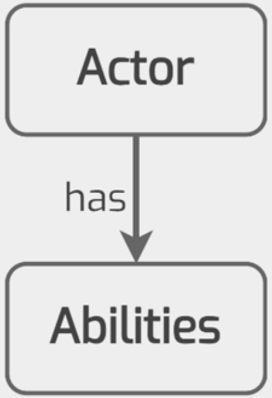
Screenplay Pattern



Screenplay Pattern

Actor

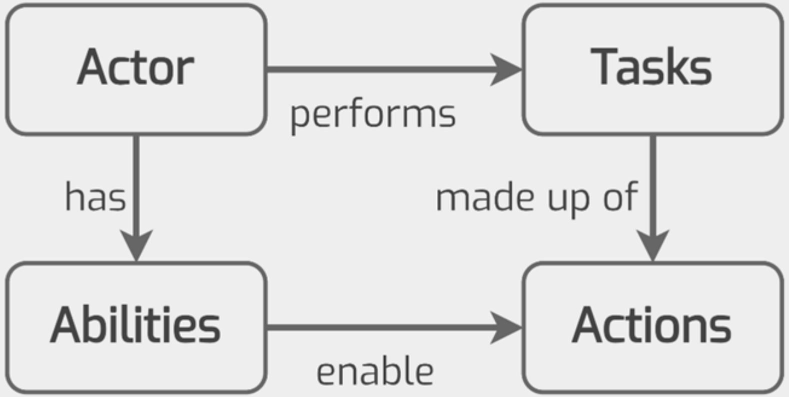
Screenplay Pattern



Screenplay Pattern

```
Actor Chriss =

        new Actor("Chriss").can(new BrowseTheWeb(BrowserType.CHROME));


WebDriver webDriver =

        chriss.uses(BrowseTheWeb.class).getWebDriver();
```

Screenplay Pattern



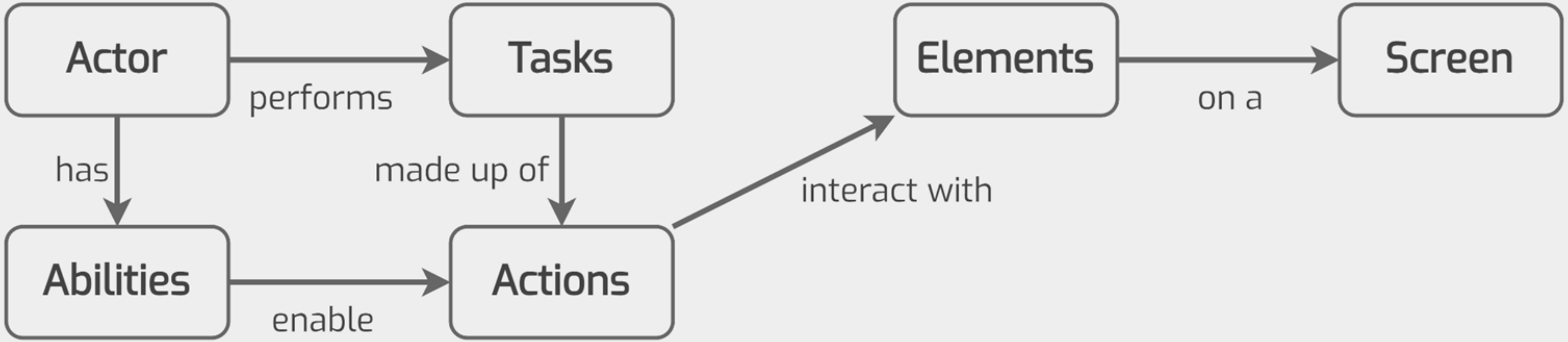
Screenplay Pattern

```
chriss.does(new Login("Chriss", "password"));

class Login implements Task {
String username;
String password;

    public void performAs(Actor actor) {
        WebDriver webDriver = actor.uses(BrowseTheWeb.class).getWebDriver();
        webDriver.get("http://parabank.parasoft.com/");
        webDriver.findElement(By.name("username")).sendKeys(username);
        webDriver.findElement(By.name("password")).sendKeys(password);
        webDriver.findElement(By.name("login")).click();
    }
}
```

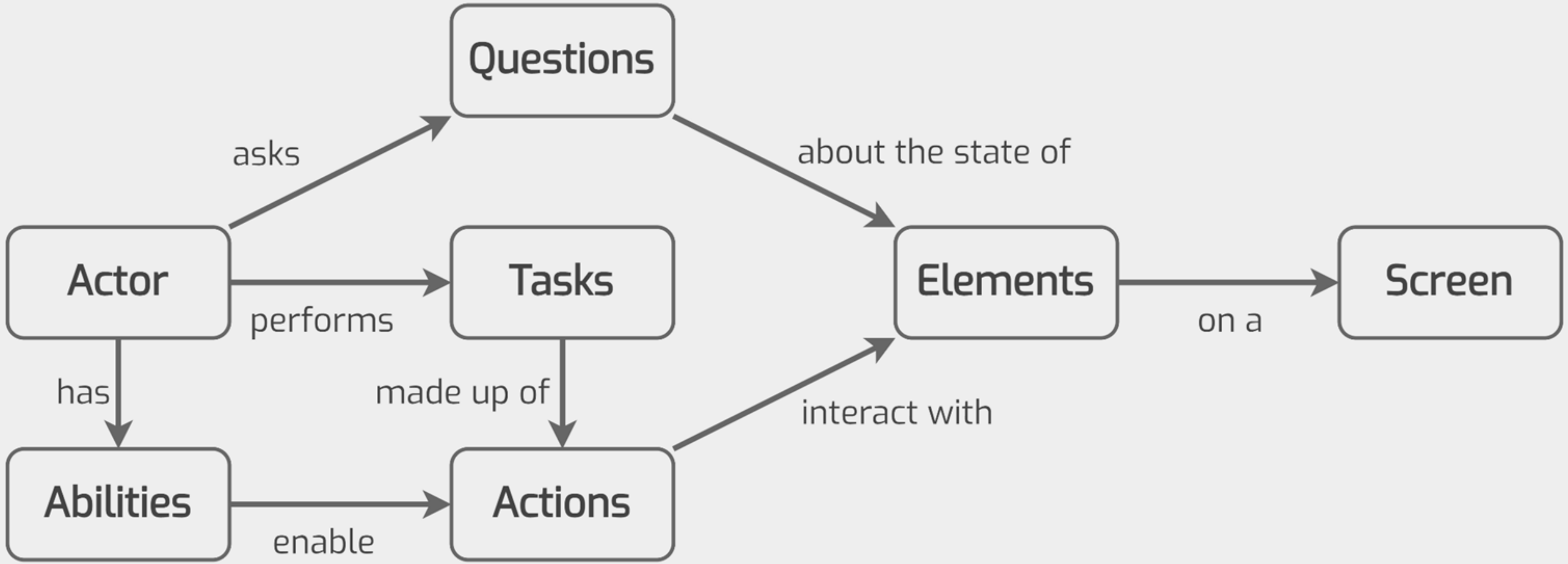
Screenplay Pattern

1. Interactions can only go into a task
2. interactions are always tied to intent and context

Screenplay Pattern



Screenplay Pattern



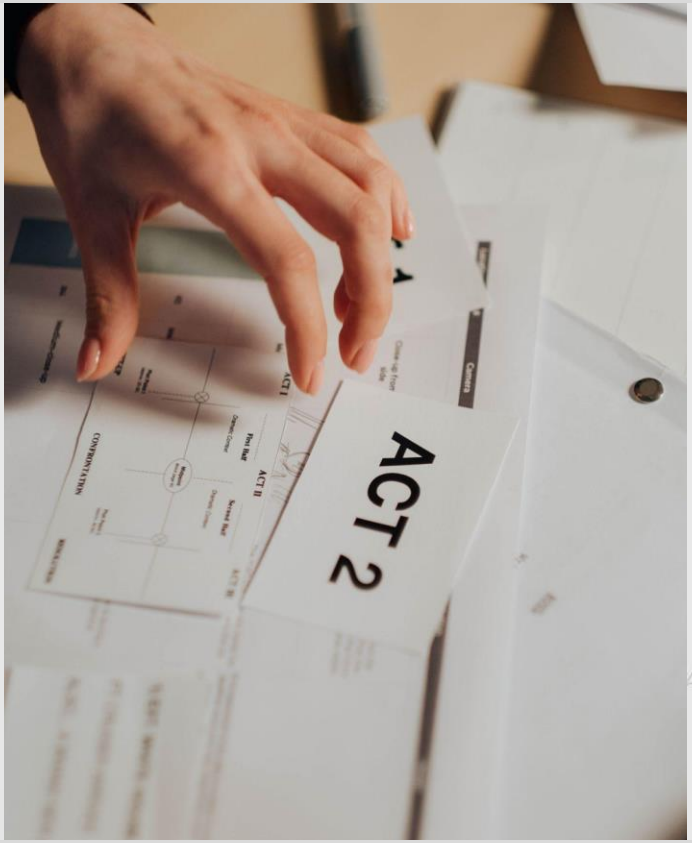
Screenplay Pattern

```
class LoggedInState implements Question<Boolean> {

    public Boolean answerAs(Actor actor) {

        final var webDriver = actor.uses(BrowseTheWeb.class).getWebDriver();

        return webDriver.findElement(By.linkText("Log Out")).isDisplayed();
    }
}
```

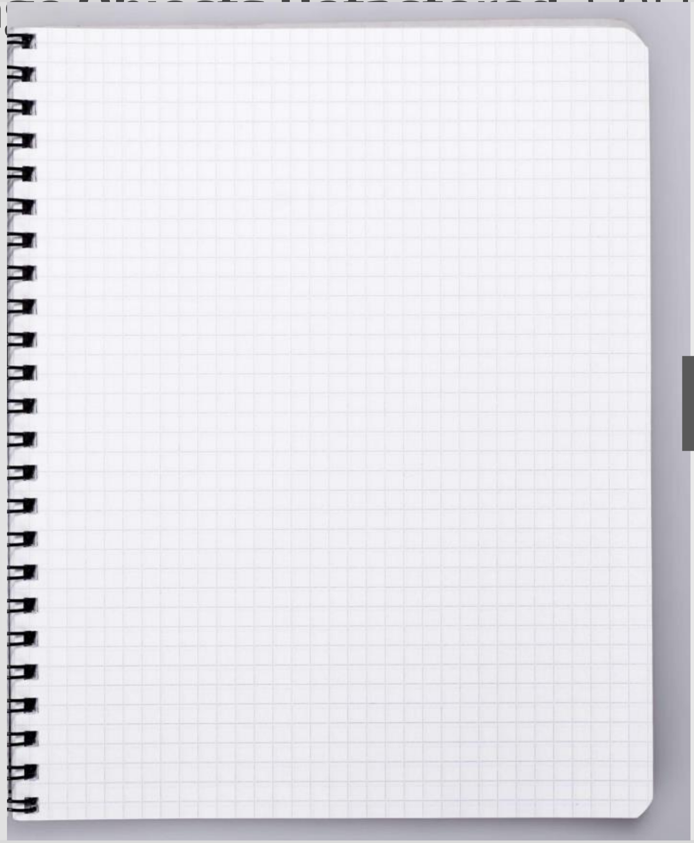
Screenplay Pattern

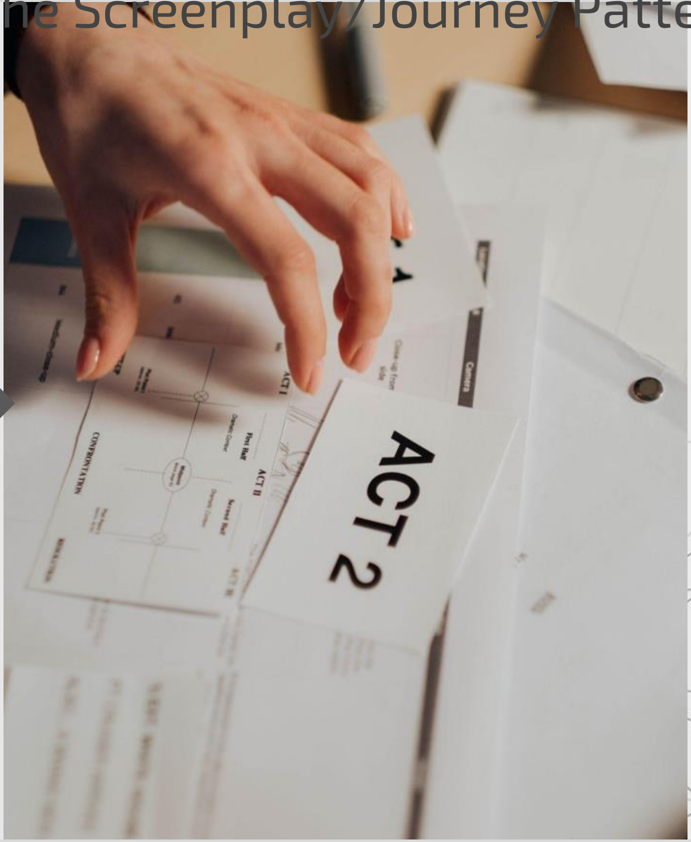
```
@Test
void register() {

    Actor chriss = new Actor("Chriss").
        can(new BrowseTheWeb(BrowserType.CHROME));

    chriss.
        does(new StartRegistration(emailAddress)).
        does(new EnterPersonalInformation(personalInformation).
        does(new EnterAddress(address).
        does(new SubmitRegisterForm());

    assertThat(romeo.checks(new LoginStatus())).isTrue();
}
```

Screenplay Pattern



Page Objects Refactored: SOLID steps to the Screenplay/Journey Pattern





How to get started

How to select a design pattern?

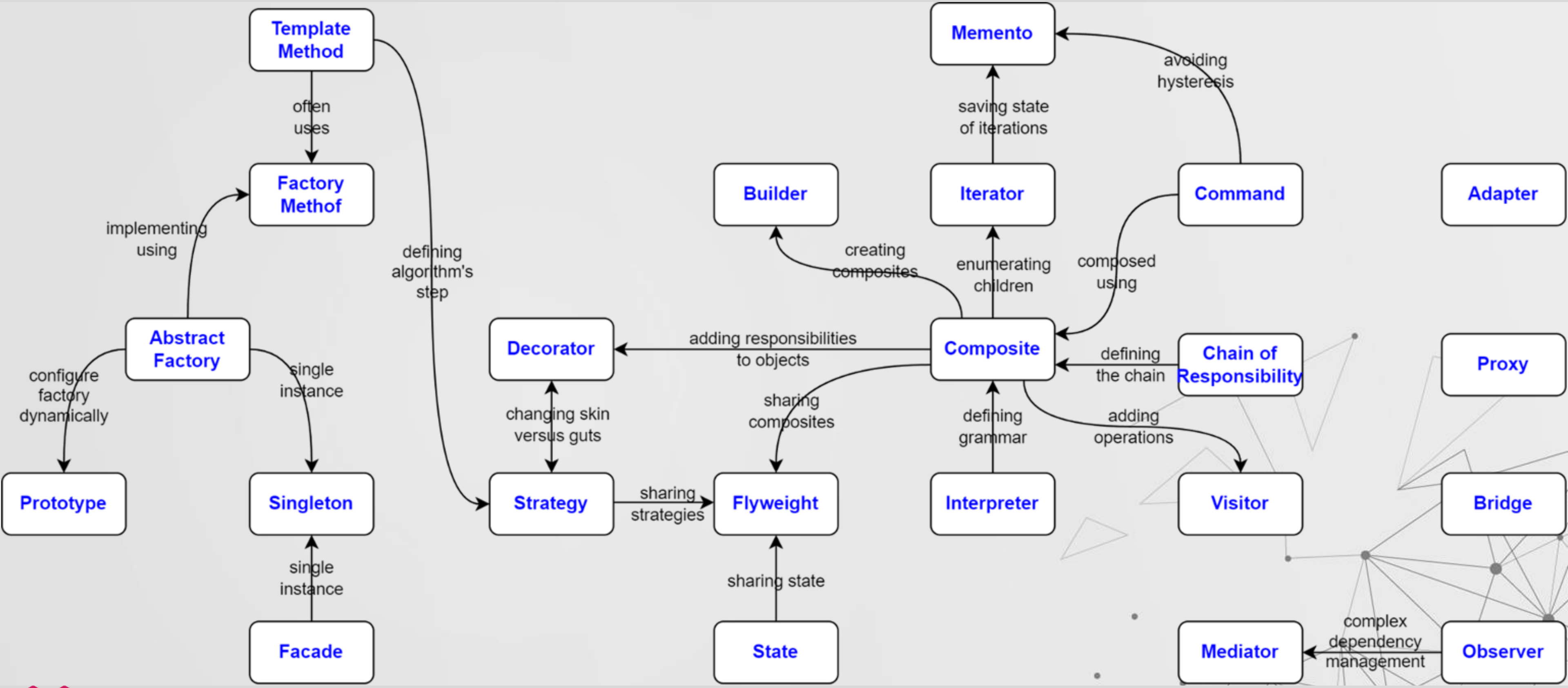
Consider how problems are solved

Scan the intents

Classification Scheme

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

Study interrelations

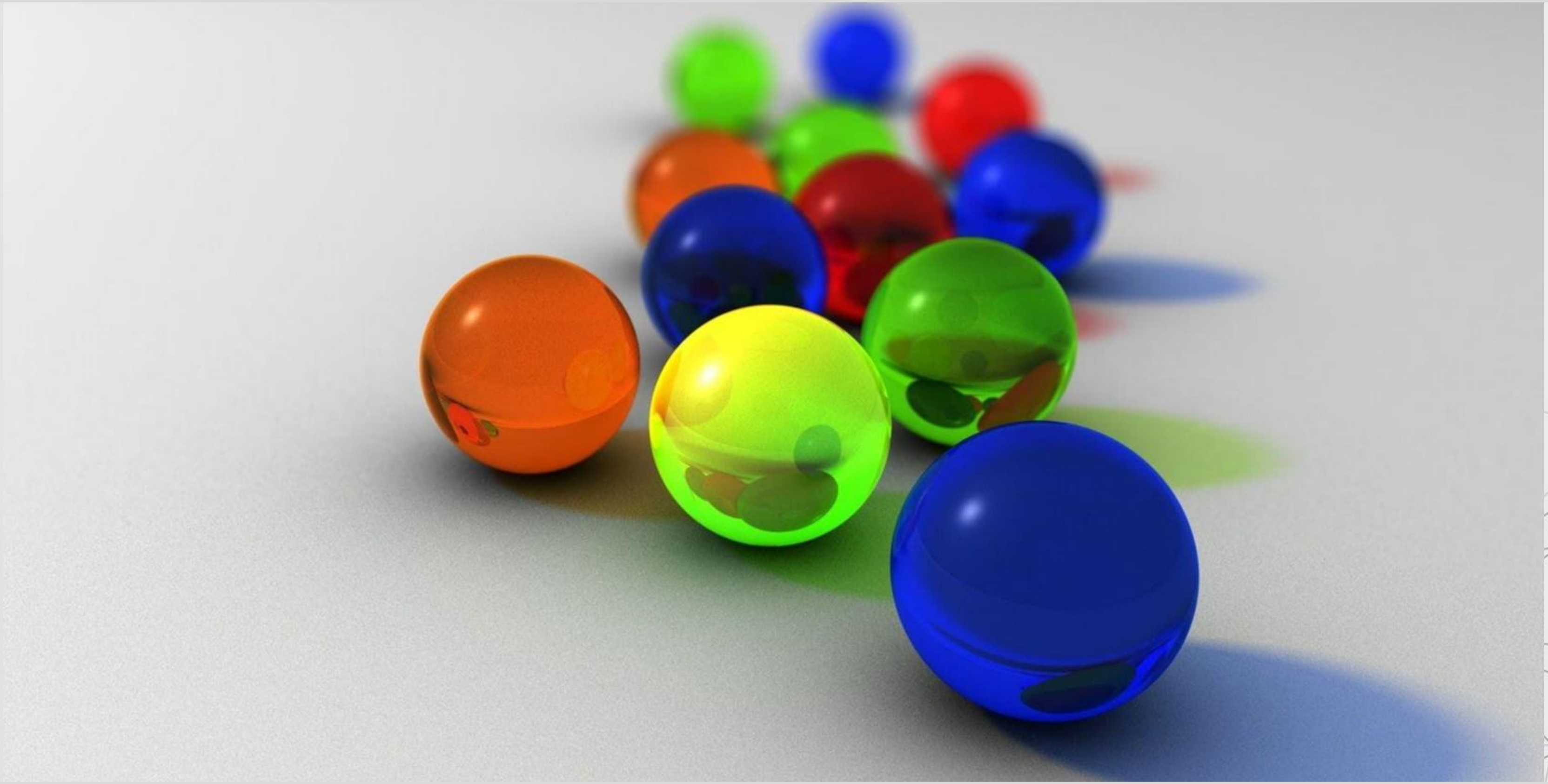


Similar purpose

Examine the cause of redesign

- Dependencies
- Algorithmic dependencies
- Tight coupling
- Extending functionality by subclassing
- Inability to alter classes conveniently

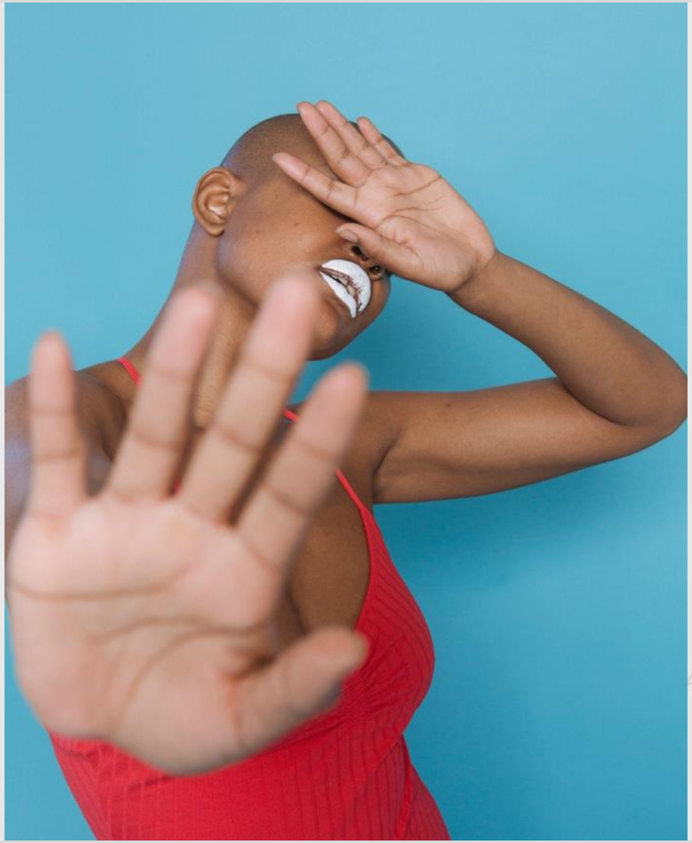
Consider what should be variable



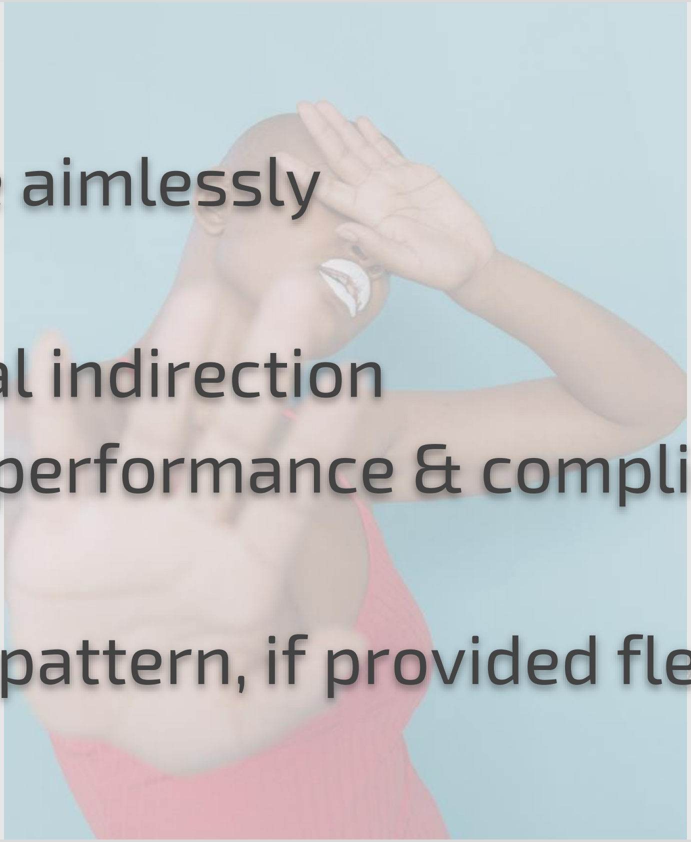
How to use a Design Pattern

- Get an overview
- Analyze structure & participants
- Study sample code
- Choose meaningful names
- Define classes
- Define specific names
- Implement operations

How not to use a Design Pattern



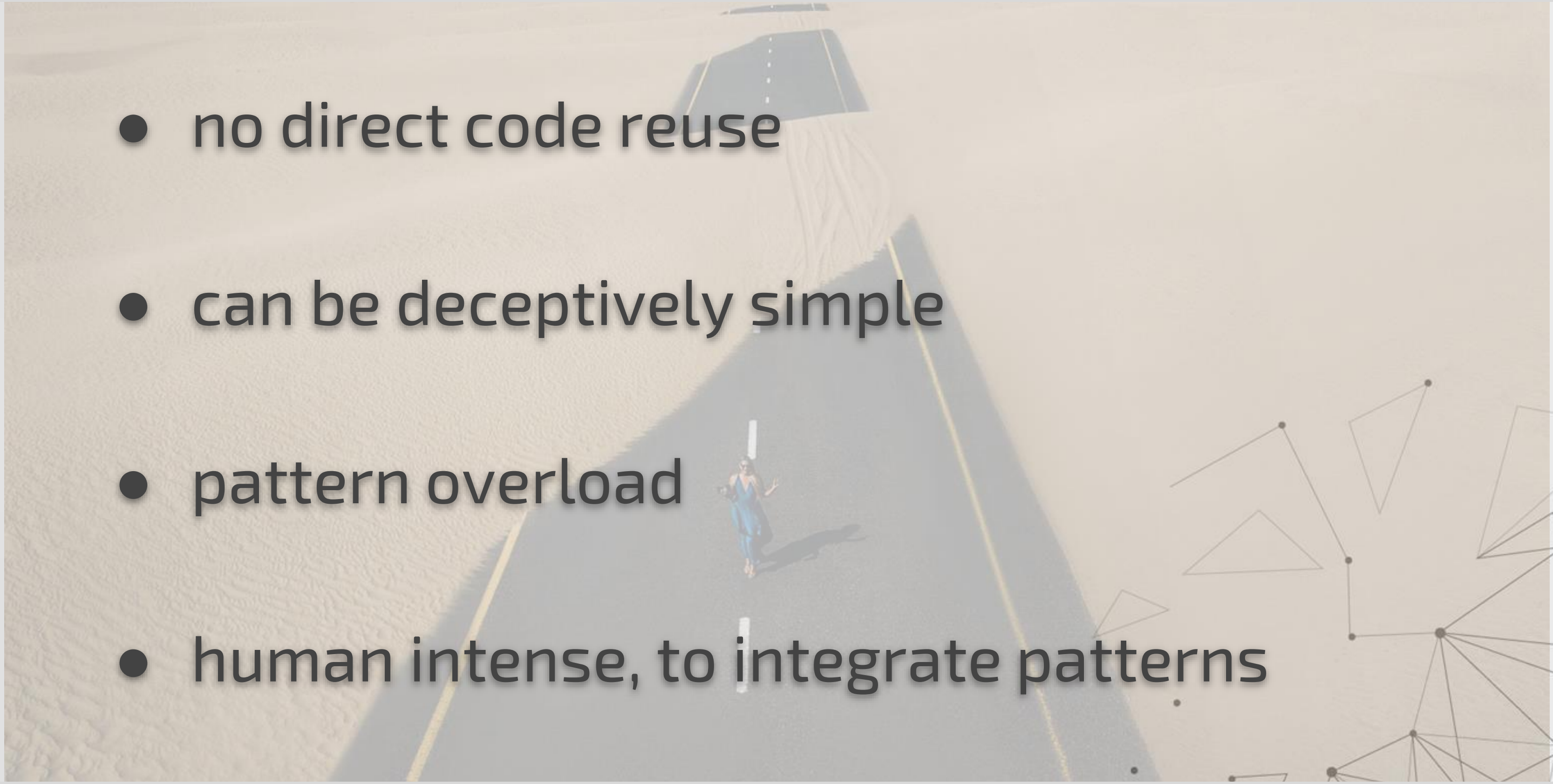
How not to use a Design Pattern



- don’t use aimlessly
- additional indirection
  → costs performance & complicates design
- only use pattern, if provided flexibility is needed

Drawbacks & Limitations

Drawbacks & Limitations



- no direct code reuse
- can be deceptively simple
- pattern overload
- human intense, to integrate patterns

Thank you!
Questions?

Ressources 1

agilitest.com/blog/writing-tests-like-shakespeare
alexilyenko.github.io/patterns-1
alexilyenko.github.io/patterns-2
alexilyenko.github.io/patterns-3
arhohuttunen.com/test-data-builders
automatetheplanet.com/advanced-page-object-pattern
automatetheplanet.com/advanced-page-object-pattern-java
automatetheplanet.com/advanced-strategy-design-pattern
automatetheplanet.com/advanced-strategy-design-pattern-in-automated-testing-java
automatetheplanet.com/decorator-design-pattern-java
automatetheplanet.com/design-patterns-for-high-quality-automated-tests-java-recording
automatetheplanet.com/facade-design-pattern
automatetheplanet.com/facade-design-pattern-java
automatetheplanet.com/fluent-page-object-pattern
automatetheplanet.com/fluent-page-object-pattern-java
automatetheplanet.com/ioc-container-page-object-pattern-steroids
automatetheplanet.com/observer-design-pattern
automatetheplanet.com/observer-design-pattern-events-delegates
automatetheplanet.com/observer-design-pattern-iobserver
automatetheplanet.com/page-object-pattern
automatetheplanet.com/page-object-pattern-java-code
automatetheplanet.com/singleton-design-pattern
automatetheplanet.com/strategy-design-pattern
automatetheplanet.com/strategy-design-pattern-java

Ressources 2

automationrhapsody.com/basic-overview-of-software-design-patterns
automationrhapsody.com/design-patterns-every-test-automation-engineer-should-know
automationrhapsody.com/facade-design-pattern
automationrhapsody.com/factory-design-pattern
automationrhapsody.com/page-objects-design-pattern
automationrhapsody.com/singleton-and-null-object-patterns
avelonpang.medium.com/gang-of-four-design-patterns-intro-e884af24b85f
blog.testproject.io/2020/06/29/design-patterns-in-test-automation/test-automation-design-patterns
bmc.com/blogs/solid-design-principles
christian-rehn.de/2009/09/03/singletons
citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.4710&rep=rep1&type=pdf
completedeveloperpodcast.com/episode-207
dev.to/thayseonofrio/how-to-get-started-with-design-patterns-iai
devbridge.com/articles/top-design-pattern-test-automation-frameworks
drive.google.com/drive/folders/14afvlixwtossryultrhbqvbpxxnnhpry
dzone.com/articles/design-patterns-in-automation-testing
dzone.com/articles/highlighting-elements-on-action-test-automation-fr
eliasnogueira.com/test-data-factory-why-and-how-to-use
fasterchaos.svbtle.com/journey-pattern
en.wikipedia.org/wiki/adapter_pattern
en.wikipedia.org/wiki/builder_pattern
en.wikipedia.org/wiki/composite_pattern
en.wikipedia.org/wiki/decorator_pattern
en.wikipedia.org/wiki/dependency_inversion_principle

Ressources 3

en.wikipedia.org/wiki/facade_pattern
en.wikipedia.org/wiki/factory_method_pattern
en.wikipedia.org/wiki/factory_method_patternattern
en.wikipedia.org/wiki/iterator_pattern
en.wikipedia.org/wiki/lazy_initialization
en.wikipedia.org/wiki/servant_(design_pattern
en.wikipedia.org/wiki/singleton_pattern
en.wikipedia.org/wiki/singleton_pattern#criticism
en.wikipedia.org/wiki/solid
en.wikipedia.org/wiki/template_method_pattern
github.com/mkutz/screenplay-workshop
ionos.com/digitalguide/websites/web-development/what-are-design-patterns
itscoderslife.wordpress.com/2019/02/05/design-patterns-importance-and-its-limitations
jasonpolites.github.io/tao-of-testing/ch3-1.1.html
kobiton.com/blog/chapter-12-test-automation-design-patterns-you-should-know
kobiton.com/book/chapter-12-test-automation-design-patterns-you-should-know
medium.com/10-minutes-qa-story/is-singleton-really-antipattern-in-test-automation-35e3b21b1f0c
ministryoftesting.com/dojo/lessons/common-ui-automation-patterns-and-methodologies-real-world-examples
muuktest.com/blog/test-design-pattern
people.cs.pitt.edu/~chang/231/seminars/s05pga/page9.htm
selenium.dev/documentation/en/guidelines_and_recommendations/page_object_models
shakespeareframework.org
slideshare.net/abagmar/perils-of-pageobject-pattern
testomat.io/blog/singleton-design-pattern-how-to-use-it-in-test-automation

Images

pexels.com/de-de/foto/arbeiten-lesen-studieren-drinnen-8086372
pexels.com/de-de/foto/fokussierte-frau-die-in-zwischenablage-schreibt-wahrend-kandidat-anstellt-5668869
pexels.com/de-de/foto/frau-die-auf-strasse-steht-2599729
pexels.com/de-de/foto/frau-hand-gesichtslos-show-6975471
pexels.com/de-de/foto/licht-haus-stapel-zuhause-7203699
pexels.com/de-de/foto/nahaufnahme-fotografie-von-wassertropfen-276502
pexels.com/de-de/foto/restaurant-blumen-bunt-stillleben-12627656
pexels.com/de-de/foto/schwarz-weiss-gestreiftes-textil-2106249
pexels.com/de-de/foto/schwarzweiss-pfeil-zeichen-6156425
pexels.com/de-de/foto/starke-sportler-die-bereit-sind-auf-dem-stadion-zu-laufen-3764011
pexels.com/de-de/foto/tilt-shift-lens-fotografie-von-funf-verschiedenen-gemusesorten-1196516
pxfuel.com/en/desktop-wallpaper-eqjhb
pxfuel.com/en/desktop-wallpaper-ilonp/download/5120x2880
pxhere.com/en/photo/1394763
pxhere.com/en/photo/539278
startertutorials.com/patterns/select-design-pattern.html/design-pattern-relationships