The Yin and Yang of Software Quality: On the Relationship between Design Patterns and Code Smells

Giammaria Giordano, Giulia Sellitto, Aurelio Sepe, Fabio Palomba, and Filomena Ferrucci

University of Salerno (Italy)
Department of Computer Science
Software Engineering (SeSa) Lab

giagiordano@unisa.it
giammariagiordano.github.io/giammaria-giordano
@giammariagiord1

Reusability Mechanisms

Reusability Mechanisms

Cost

Reusability Mechanisms

- Cost
- Time

Reusability Mechanisms

- Cost
- Time
- Effort

Reusability Mechanisms

- Legacy System Wrapping
- Design Patterns
- Service Oriented Systems
- Third-Party Libraries
- Programming Abstraction
- Program Generator

Reusability Mechanisms

Legacy System Wrapping

Third-Party Libraries

Reusability Mechanisms are **essential** during **Software Evolution** and **Maintenance!**

Service Oriented Systems

Program Generator

Design Patterns

Reusable solutions for **common problems** that arise **during** the **design** and **development** of **software**

Code Smells

A **symptom** of **poor design** that **can lead** to **increased effort** during **maintenance** and **evolution activities**

State of the Art

Myth or Reality? Analyzing the Effect of D Patterns on Software Maintainability

Péter Hegedűs, Dénes Bán, Rudolf Ferenc, and Tibor Gyimóth

University of Szeged, Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{hpeter,zealot,ferenc,gyimothy}@inf.u-szeged.hu

Abstract. Although the belief of utilizing design patterns to create b ter quality software is fairly widespread, there is relatively little resear objectively indicating that their usage is indeed beneficial.
In this paper we try to reveal the connection between design patte and software maintainability. We analyzed more than 300 revisions JHotDraw, a Java GUI framework whose design relies heavily on so well-known design patterns. We used our probabilistic quality model estimating the maintainability and we parsed the javadoc annotations the source code for gathering the pattern instances.
We found that every introduced pattern instance caused an impro ment in the different quality attributes. Moreover, the average desi pattern line density showed a very high, 0.89 Pearson correlation w the estimated maintainability values. Although the amount of availa empirical data is still very small, these first results suggest that the usa of design patterns do improve code maintainability.

Keywords: Design patterns, Software maintainability, Empirical va dation, OO design

1 Introduction

Since their introduction by Gamma et al. [7], there has been a growing i the use of design patterns. Object-Oriented (OO) design patterns repres known solutions to common design problems in a given context. The belief is that applying design patterns results in a better OO design, they improve software quality as well [7, 16].

However, there is a little empirical evidence that design patterns r prove code quality. Moreover, some studies suggest that the use of design not necessarily result in good design [13,20]. The problem of empirical v is that it is very hard to assess the effect of design patterns to high lev characteristics e.g.: maintainability, reusability, understandability, etc. some approaches that manually evaluate the impact of certain design on different quality attributes [11].

We also try to reveal the connection between design patterns and quality but we focus on the maintainability of the source code. As m crete maintainability models exist (e.g. [2, 4, 8]) we could choose a mo

A Controlled Experiment Comparing the Maintainability of Programs Designed wit without Design Patterns—A Replication i Programming Environment

MAREK VOKÁČ
Simula Research Laboratory, N-1325, Lysaker, Norway

WALTER TICHY
Universität Karlsruhe, Postfach 6980, D-76128 Karlsruhe, Germany

DAG I. K. SJØBERG
Simula Research Laboratory, N-1325, Lysaker, Norway

ERIK ARISHOLM
Simula Research Laboratory, N-1325, Lysaker, Norway

MAGNE ALDRIN
Norwegian Computing Center, P.O. Box 114 Blindern, N-0314 Oslo, Norway

Editor: Dieter Rombach

Abstract. Software "design patterns" seek to package proven solutions to design pr makes it possible to find, adapt and reuse them. To support the industrial use o research investigates when, and how, using patterns is beneficial, and whether so difficult to use than others. This paper describes a replication of an earlier controlled patterns in maintenance, with major extensions. Experimental realism was incr programming environment instead of pen and paper, and paid professionals consultancy companies as subjects.

Measurements of elapsed time and correctness were analyzed using regression mo method that took into account the correlations present in the raw data. Together the subjects' work, this made possible a better qualitative understanding of the res

The results indicate quite strongly that some patterns are much easier to und others. In particular, the Visitor pattern caused much confusion. Conversely, the pat a certain extent, Decorator were grasped and used intuitively, even by subjects with of patterns.

The implication is that design patterns are not universally good or bad, but must matches the problem and the people. When approaching a program with docum even basic training can improve both the speed and quality of maintenance activit

Keywords: Controlled experiment, design patterns, real programming environment

Do Design Patterns Impact Software Quality Positively?

Foutse Khomh* and Yann-Gaël Guéhéneuc
Ptidej Team, GEODES, DIRO, University of Montreal,
C.P. 6128 succursale Centre Ville Montréal, Quebec, H3C 3J7, Canada

E-mail: {foutsekh,guehene}@iro.umontreal.ca

Abstract

We study the impact of design patterns on quality attributes in the context of software maintenance and evolution. We show that, contrary to popular beliefs, design patterns in practice *impact negatively several quality attributes, thus providing concrete evidence against common lore. We then study design patterns and object-oriented best practices by formulating a second hypothesis on the impact of these principles on quality. We show that results for some design patterns cannot be explained and conclude on the need for further studies. Thus, we bring further evidence that design patterns should be used with caution during development because they may actually impede maintenance and evolution.*

1. Introduction

Many studies in the literature (including some by these authors) have for premise that design patterns [2] improve the quality of object-oriented software systems, because design patterns are supposed to improve the quality of systems, for example [2, page xiii] or [10].

Yet, some studies, *e.g.*, [11], suggest that the use of design patterns do not always result in "good" designs. For example, a tangled implementation of patterns impacts negatively quality [8]. Also, patterns generally ease future enhancement at the expense of simplicity.

There is little empirical evidence to support the claims of improved reusability[1], expandability and understandability as put forward in [2] when applying design patterns.

Therefore, we carry an empirical study of the impact of design patterns on the quality of systems as perceived by software engineers in the context of maintenance and evolution. Our hypothesis verifies software engineering lore: design patterns impact softwa ity positively. Our objective is to provide evi confirm or refute the hypothesis. We perform t by asking respondents their evaluations of the of design patterns on quality after their use.

We present detailed results for three des terns: Abstract Factory, Composite, Flywei three quality attributes: reusability, underst ity, and expandability. Results for other patt quality attributes can be found in [5]. We sh contrary to popular beliefs, patterns *in practic* always improve quality attributes, thus provi idence against common lore. We attempt to these results using object-oriented best practi conclude on the need for further studies and t terns should be used with caution because t actually impede maintenance and evolution.

Section 2 presents related work and their tions. Section 3 states the hypothesis and o of the study and presents our data collection cessing. Section 4 describes our quantification and presents the results of our survey. Sectio tains a discussion of the results. Section 6 c our research, discusses the threats to the validi study and introduces future work.

2. Related Work

Since their introduction by Gamma et al. [2] there has been a growing interest on the use patterns. We present here some lines of work impact of patterns on quality.

Lange and Nakamura demonstrated [6] tha patterns can serve as guide in program explora thus make the process of program understandi efficient. However this study was limited to quality attribute and to a little number of pat

Wydaeghe et al. [12] presented a study on crete use of six design patterns. They discu impact of these patterns on reusability, mo

[1] Although reusability in [2] may refer to the reusability of the solutions of the design patterns, we consider reusability as the reusability of the piece of code in which a pattern is implemented.

An exploratory study of the impact of antipatterns on class change- and fault-proneness

Foutse Khomh · Massimiliano Di Penta · Yann-Gaël Guéhéneuc · Giuliano Antoniol

Published online: 6 August 2011

Editor: Jim Whitehead

Abstract Antipatterns are poor design choices that are conjectured to make object-oriented systems harder to maintain. We investigate the impact of antipatterns on classes in object-oriented systems by studying the relation between the presence of antipatterns and the change- and fault-proneness of the classes. We detect 13 antipatterns in 54 releases of ArgoUML, Eclipse, Mylyn, and Rhino, and analyse (1) to what extent classes participating in antipatterns have higher odds to change or to be subject to fault-fixing than other classes, (2) to what extent these odds (if higher) are due to the sizes of the classes or to the presence of antipatterns, and (3) what kinds of changes affect classes participating in antipatterns. We show that, in almost all releases of the four systems, classes participating in antipatterns are more change- and fault-prone than others. We also show that size alone cannot explain the higher odds of classes with antipatterns to underwent a (fault-fixing) change than other

We thank Marc Eaddy for making his data on faults freely available. This work has been partly funded by the NSERC Research Chairs in Software Change and Evolution and in Software Patterns and Patterns of Software.

F. Khomh (✉)
Department of Electrical and Computer Engineering,
Queen's University, Kingston, ON, Canada
e-mail: foutse.khomh@queensu.ca

M. D. Penta
Department of Engineering, University of Sannio, Benevento, Italy
e-mail: dipenta@unisannio.it

Y.-G. Guéhéneuc · G. Antoniol
SOCCER Lab. and Ptidej Team, Département de Génie Informatique et Génie Logiciel,
École Polytechnique de Montréal, Montréal, QC, Canada

Y.-G. Guéhéneuc
e-mail: yann-gael.gueheneuc@polymtl.ca

G. Antoniol
e-mail: antoniol@ieee.org

State of the Art

Most of previous work **investigated** on the relationship between **Design Patterns** and **Code Quality without** taking into account **Code Smells!**

State of the Art

The relationship between design patterns and code smells: An exploratory study

Bartosz Walter[a,*], Tarek Alkhaeir[b]

[a] Faculty of Computing, Poznań University of Technology, Poznań, Poland
[b] Poznań Supercomputing and Networking Center, Poznań, Poland

ARTICLE INFO

Article history:
Received 29 April 2015
Revised 14 February 2016
Accepted 14 February 2016
Available online 3 March 2016

Keywords:
Design patterns
Code smells
Software evolution
Empirical study

ABSTRACT

Context—Design patterns represent recommended generic solutions to various design problems, whereas code smells are symptoms of design issues that could hinder further maintenance of a software system. We can intuitively expect that both concepts are mutually exclusive, and the presence of patterns is correlated with the absence of code smells. However, the existing experimental evidence supporting this claim is still insufficient, and studies separately analyzing the impact of smells and patterns on code quality deliver diverse results.

Objective—The aim of the paper is threefold: (1) to determine if and how the presence of the design patterns is linked to the presence of code smells, (2) to investigate if and how these relationships change throughout evolution of code, and (3) to identify the relationships between individual patterns and code smells.

Method—We analyze nine design patterns and seven code smells in two medium-size, long-evolving, open source Java systems. In particular, we explore how the presence of design patterns impacts the presence of code smells, analyze if this link evolves over time, and extract association rules that describe their individual relationships.

Results—Classes participating in design patterns appear to display code smells less frequently than other classes. The observed effect is stronger for some patterns (e.g., Singleton, State-Strategy) and weaker for others (e.g., Composite). The ratio between the relative number of smells in the classes participating in patterns and the relative number of smells in other classes, is approximately stable or slightly decreasing in time.

Conclusion—This observation could be used to anticipate the smell-proneness of individual classes, and improve code smell detectors. Overall, our findings indicate that the presence of design patterns is linked with a lower number of code smell instances. This could support programmers in a context-sensitive analysis of smells in code.

1. Introduction

Design patterns and code smells represent two different approaches to assuring source code quality. The first approach, perfective, is focused on solutions which positively impact some attributes of quality, and which have been empirically validated. The other approach, preventive, concentrates on detecting and removing elements that could be harmful for a software system, or make it insufficiently effective. Moreover, the preventive methods also include mechanisms that can identify symptoms of anomalies before their negative impact on quality grows and could become destructive for the system.

Design patterns represent the perfective group as they describe practically validated solutions to recurring design problems. They can easily be adapted and applied several times without changing the core of the concept. Since their introduction to software engineering by the Gang of Four in 1994 [15], they have been an object of rising interest of programmers and researchers, and practically demonstrated their ability to be implemented in different contexts. Intuitively, it is expected that the use of design patterns

* Corresponding author. Tel.: +48 616652980.
E-mail addresses: bartosz.walter@cs.put.poznan.pl (B. Walter), tarekkh@man.poznan.pl (T. Alkhaeir).

http://dx.doi.org/10.1016/j.infsof.2016.02.003
0950-5849/

State of the Art

erns and code smells: An

CrossMark

rns represent recommended generic solutions to various design problems, whereas
toms of design issues that could hinder further maintenance of a software system.
pect that both concepts are mutually exclusive, and the presence of patterns is
sence of code smells. However, the existing experimental evidence supporting this
ent, and studies separately analyzing the impact of smells and patterns on code
results.

f the paper is threefold: (1) to determine if and how the presence of the design
he presence of code smells, (2) to investigate if and how these relationships change
of code, and (3) to identify the relationships between individual patterns and code

nine design patterns and seven code smells in two medium-size, long-evolving,
ems. In particular, we explore how the presence of design patterns impacts the
ls, analyze if this link evolves over time, and extract association rules that describe
nships.

ipating in design patterns appear to display code smells less frequently than other
effect is stronger for some patterns (e.g., Singleton, State-Strategy) and weaker for
e). The ratio between the relative number of smells in the classes participating in
ve number of smells in other classes, is approximately stable or slightly decreasing

rvation could be used to anticipate the smell-proneness of individual classes, and
tectors. Overall, our findings indicate that the presence of design patterns is linked
of code smell instances. This could support programmers in a context-sensitive
ode.

removing elements that could be harmful for a software system, or make it insufficiently effective. Moreover, the preventive methods also include mechanisms that can identify symptoms of anomalies before their negative impact on quality grows and could become destructive for the system.

Design patterns represent the perfective group as they describe practically validated solutions to recurring design problems. They can easily be adapted and applied several times without changing the core of the concept. Since their introduction to software engineering by the Gang of Four in 1994 [15], they have been an object of rising interest of programmers and researchers, and practically demonstrated their ability to be implemented in different contexts. Intuitively, it is expected that the use of design patterns

Two medium projects have been analyzed from an **evolutionary** standpoint

State of the Art

erns and code smells: An

CrossMark

rns represent recommended generic solutions to various design problems, whereas
toms of design issues that could hinder further maintenance of a software system.
pect that both concepts are mutually exclusive, and the presence of patterns is
sence of code smells. However, the existing experimental evidence supporting this
ent, and studies separately analyzing the impact of smells and patterns on code
results.

f the paper is threefold: (1) to determine if and how the presence of the design
he presence of code smells, (2) to investigate if and how these relationships change
of code, and (3) to identify the relationships between individual patterns and code

nine design patterns and seven code smells in two medium-size, long-evolving,
ems. In particular, we explore how the presence of design patterns impacts the
ls, analyze if this link evolves over time, and extract association rules that describe
nships.

ipating in design patterns appear to display code smells less frequently than other
effect is stronger for some patterns (e.g., Singleton, State-Strategy) and weaker for
e). The ratio between the relative number of smells in the classes participating in
ve number of smells in other classes, is approximately stable or slightly decreasing

ervation could be used to anticipate the smell-proneness of individual classes, and
etectors. Overall, our findings indicate that the presence of design patterns is linked
of code smell instances. This could support programmers in a context-sensitive
ode.

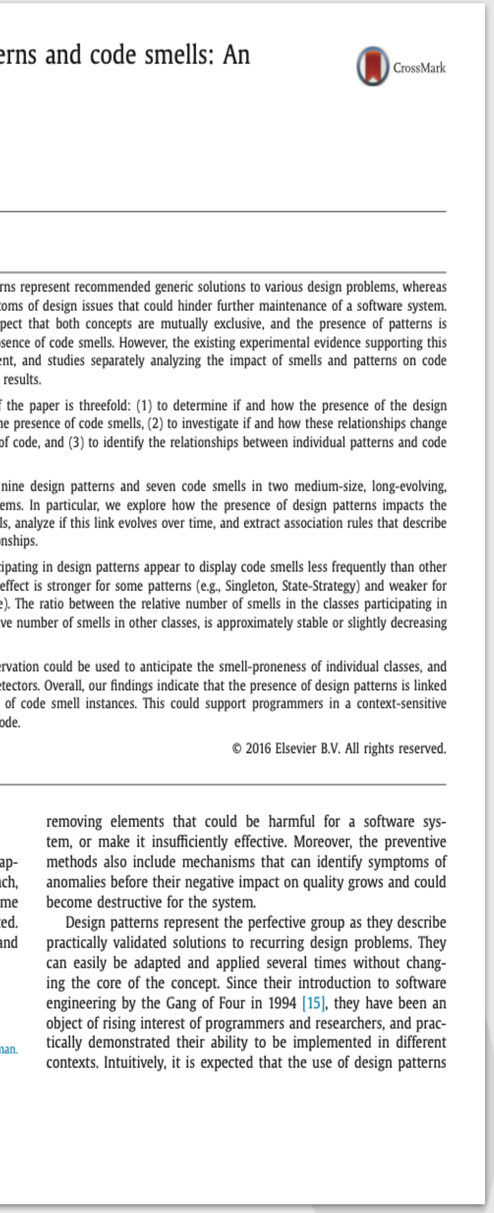
removing elements that could be harmful for a software system, or make it insufficiently effective. Moreover, the preventive methods also include mechanisms that can identify symptoms of anomalies before their negative impact on quality grows and could become destructive for the system.

Design patterns represent the perfective group as they describe practically validated solutions to recurring design problems. They can easily be adapted and applied several times without changing the core of the concept. Since their introduction to software engineering by the Gang of Four in 1994 [15], they have been an object of rising interest of programmers and researchers, and practically demonstrated their ability to be implemented in different contexts. Intuitively, it is expected that the use of design patterns

Two medium projects have been analyzed from an **evolutionary** standpoint

The authors consider 7 kinds of code smells principally related to **Coupling**

State of the Art



erns and code smells: An

CrossMark

rns represent recommended generic solutions to various design problems, whereas
toms of design issues that could hinder further maintenance of a software system.
pect that both concepts are mutually exclusive, and the presence of patterns is
sence of code smells. However, the existing experimental evidence supporting this
ent, and studies separately analyzing the impact of smells and patterns on code
results.

f the paper is threefold: (1) to determine if and how the presence of the design
he presence of code smells, (2) to investigate if and how these relationships change
of code, and (3) to identify the relationships between individual patterns and code

nine design patterns and seven code smells in two medium-size, long-evolving,
ems. In particular, we explore how the presence of design patterns impacts the
ls, analyze if this link evolves over time, and extract association rules that describe
nships.

ipating in design patterns appear to display code smells less frequently than other
effect is stronger for some patterns (e.g., Singleton, State-Strategy) and weaker for
e). The ratio between the relative number of smells in the classes participating in
ve number of smells in other classes, is approximately stable or slightly decreasing

rvation could be used to anticipate the smell-proneness of individual classes, and
tectors. Overall, our findings indicate that the presence of design patterns is linked
of code smell instances. This could support programmers in a context-sensitive
ode.

removing elements that could be harmful for a software system, or make it insufficiently effective. Moreover, the preventive methods also include mechanisms that can identify symptoms of anomalies before their negative impact on quality grows and could become destructive for the system.

Design patterns represent the perfective group as they describe practically validated solutions to recurring design problems. They can easily be adapted and applied several times without changing the core of the concept. Since their introduction to software engineering by the Gang of Four in 1994 [15], they have been an object of rising interest of programmers and researchers, and practically demonstrated their ability to be implemented in different contexts. Intuitively, it is expected that the use of design patterns

Two medium projects have been analyzed from an **evolutionary** standpoint

The authors consider 7 kinds of code smells principally related to **Coupling**

The ratio between the **relative number** of **Smells** in **classes participating** in **Design Patterns** and the relative **number** of **Smells** in **other classes** is **approximately the same**

Limitations

Limitations

The paper considers **only two medium** size projects

Limitations

The paper considers **only two medium** size projects

No investigation on the relationship between **Design Patterns** and **Code Smells** that **impact Understandability** and **Code**

Why is it important?

Design Patterns are commonly used facilitate the **Maintenance** of source code

For this reason, it is **essential** to understand the possibile **relationship** with specific **Code Smells** that can impact the **Understandability** and **Code Comprehension**

Goal

Investigating **whether** and **how Design Patterns instances** are **related** to the **emergence** of **Code Smells instances** that impact **Understandability** and **Code Comprehension**

How do we assessed our goal?

We conducted a **large empirical investigation** by considering **over 540 releases** of **15 projects**, on the **relationship** between
Design Patterns and **Code Smells**

Research Questions

Research Questions

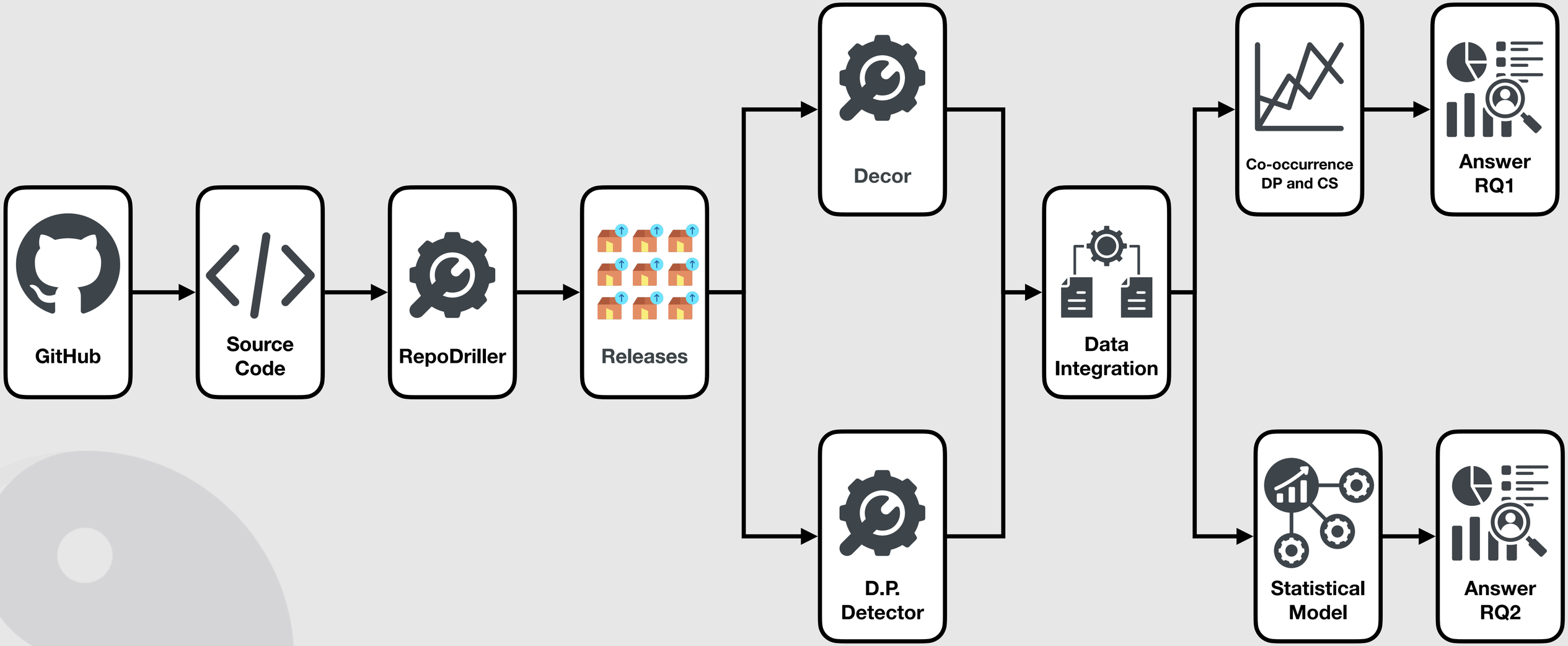
What are the **co-occurrences** in terms of classes between **Design Patterns** and **Code Smells**?

Research Questions

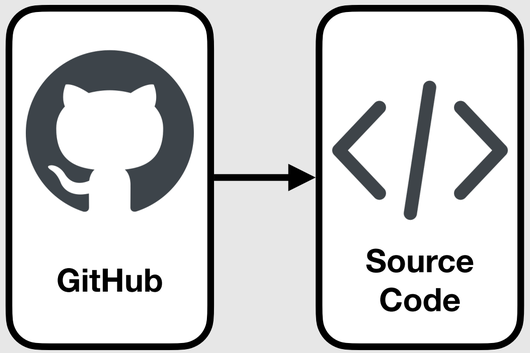
What are the **co-occurrences** in terms of classes between **Design Patterns** and **Code Smells**?

To what extent does the **presence** of **Design Patterns affect Code Smells**?

Research Process

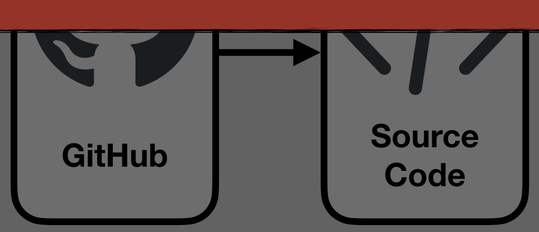


Research Process



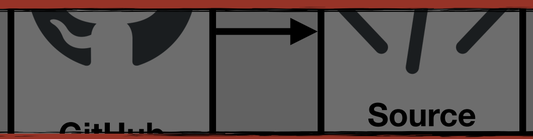
Research Process

We mined **15** popular **Java projects**



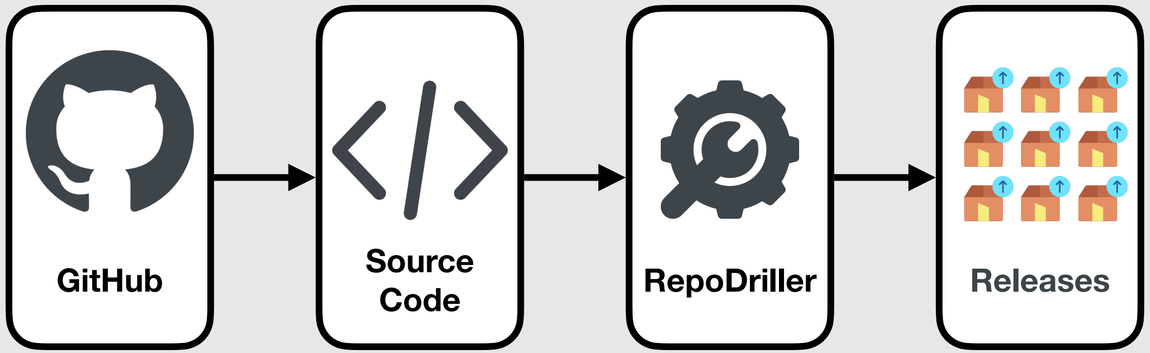
Research Process

We mined **15** popular **Java projects**



We **selected** projects with **at least one Design Pattern** and based on the **popularity** in terms of **number of stars**

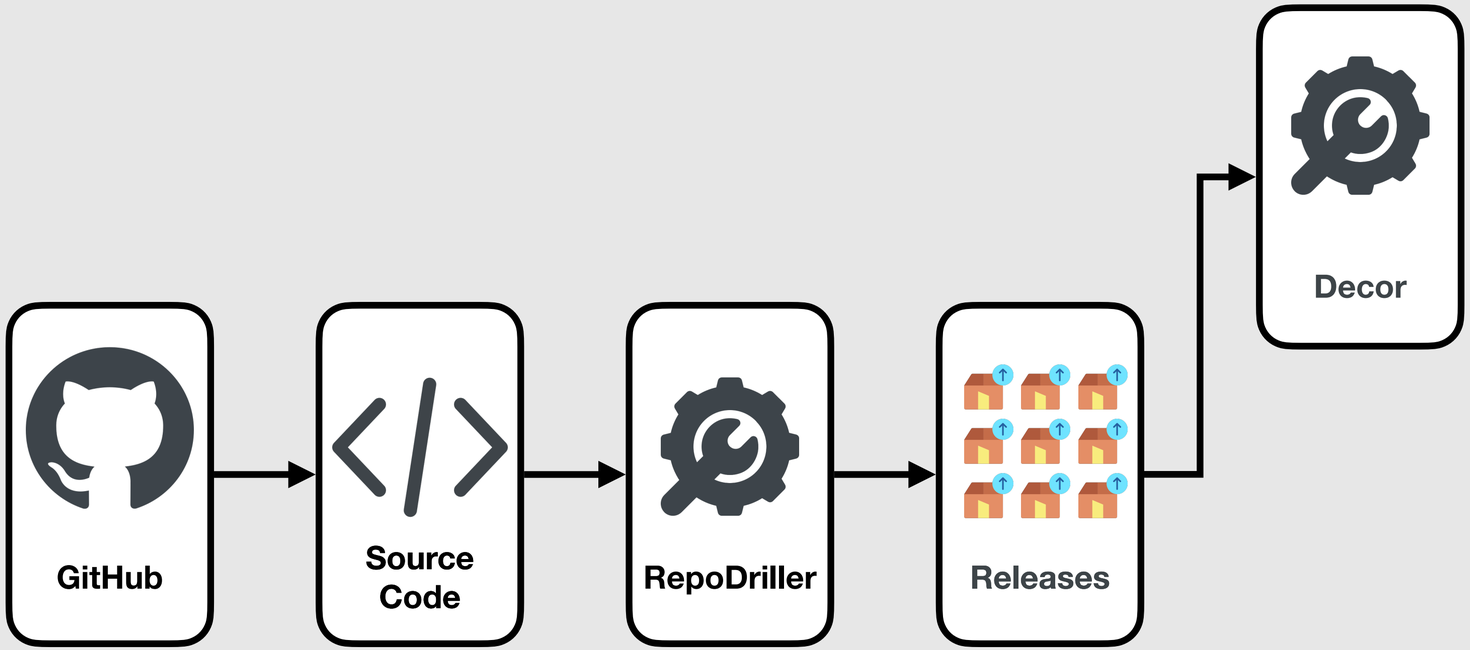
Research Process



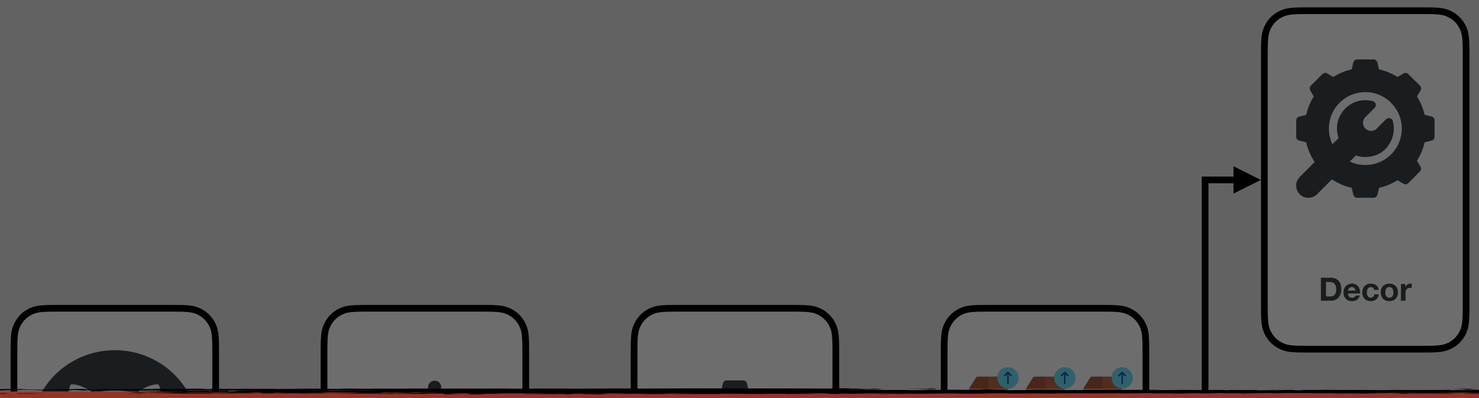
Research Process

We used as **experimental objects over 540 releases**

Research Process

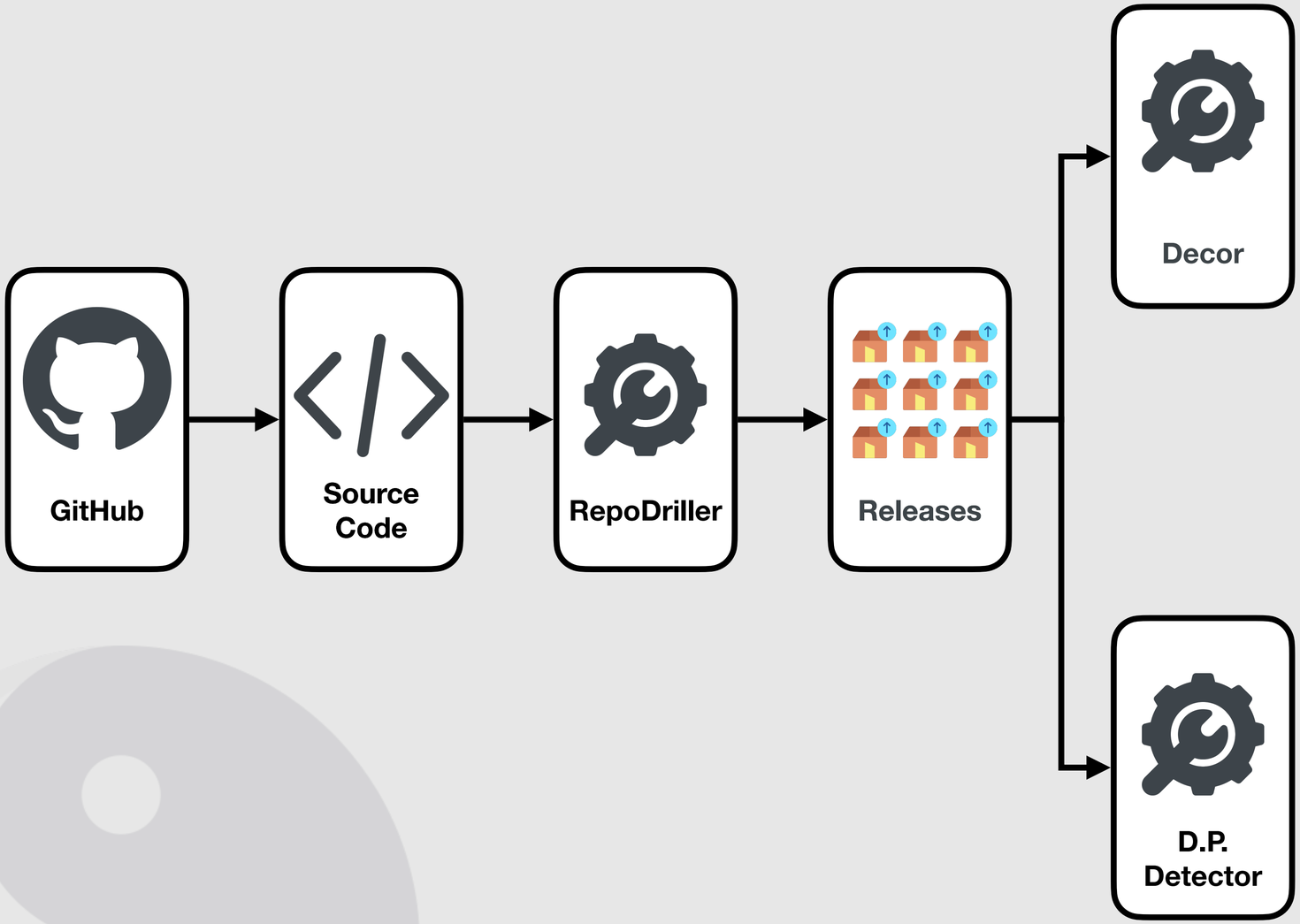


Research Process



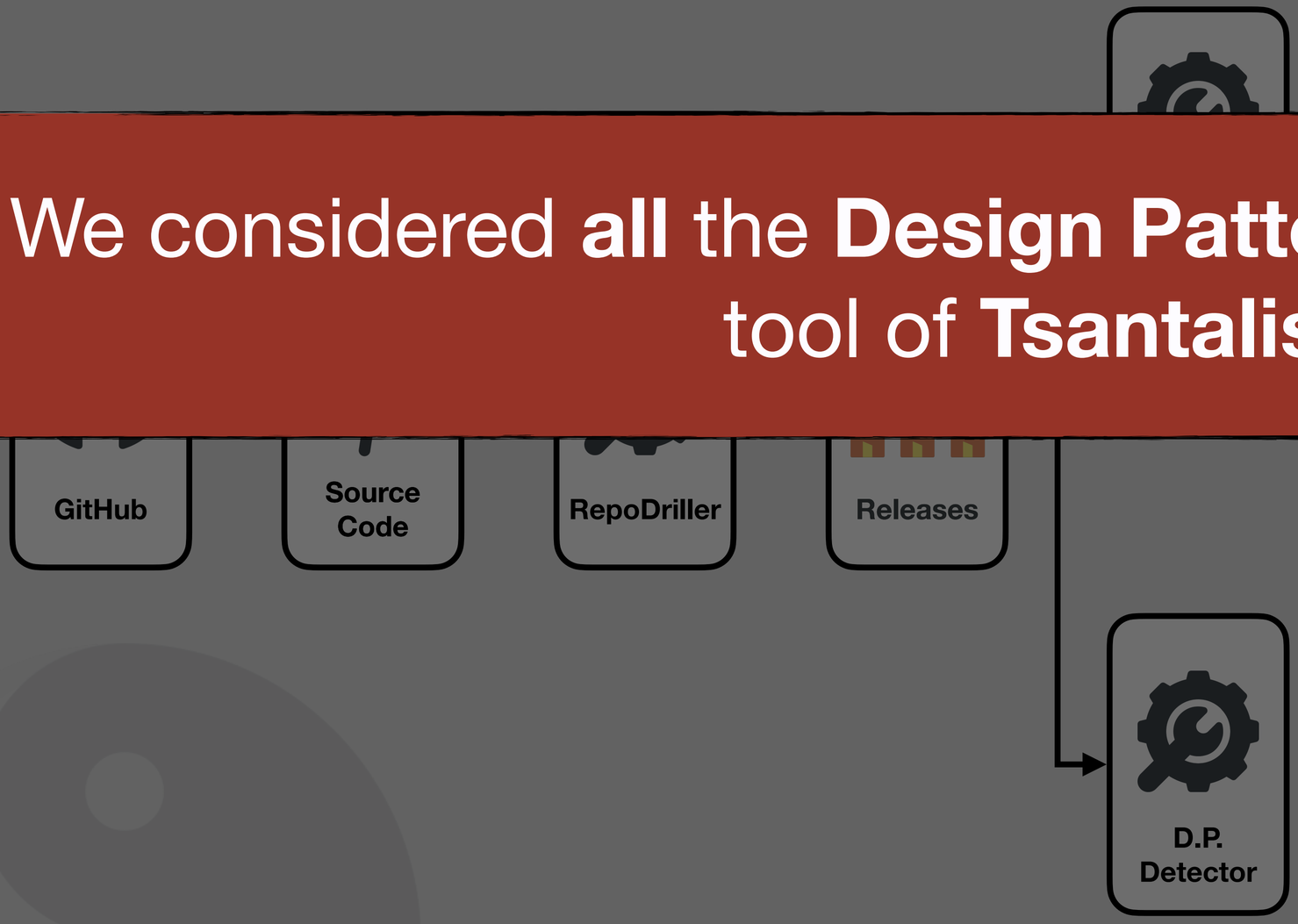
We set up Decor to extract Smells that impact **Understandability** and **Code Comprehension** i.e., **Complex Class**, **God Class**, and **Spaghetti Code**

Research Process

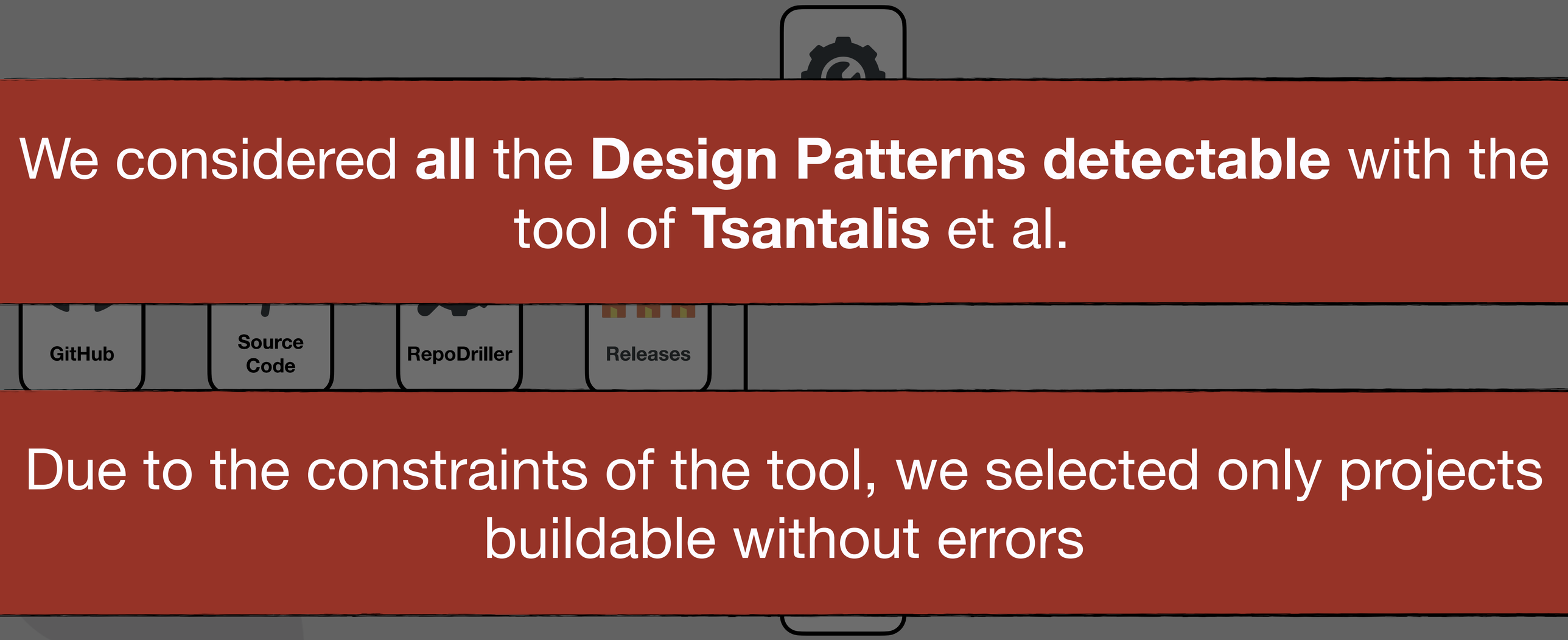


Research Process

We considered **all** the **Design Patterns detectable** with the tool of **Tsantalis** et al.



Research Process



We considered **all** the **Design Patterns detectable** with the tool of **Tsantalis** et al.

Due to the constraints of the tool, we selected only projects buildable without errors

List of Design Patterns

- Adapter/Command
- Bridge
- Singleton
- Template Method
- Proxy
- State/Strategy
- Decorator
- Factory Method
- Component
- Observer

List of Design Patterns

Adapter/Command

State/Strategy

Due to the identical UML, the tool is not able to detect differences between “Adapter” and “Command” and “State” and “Strategy”

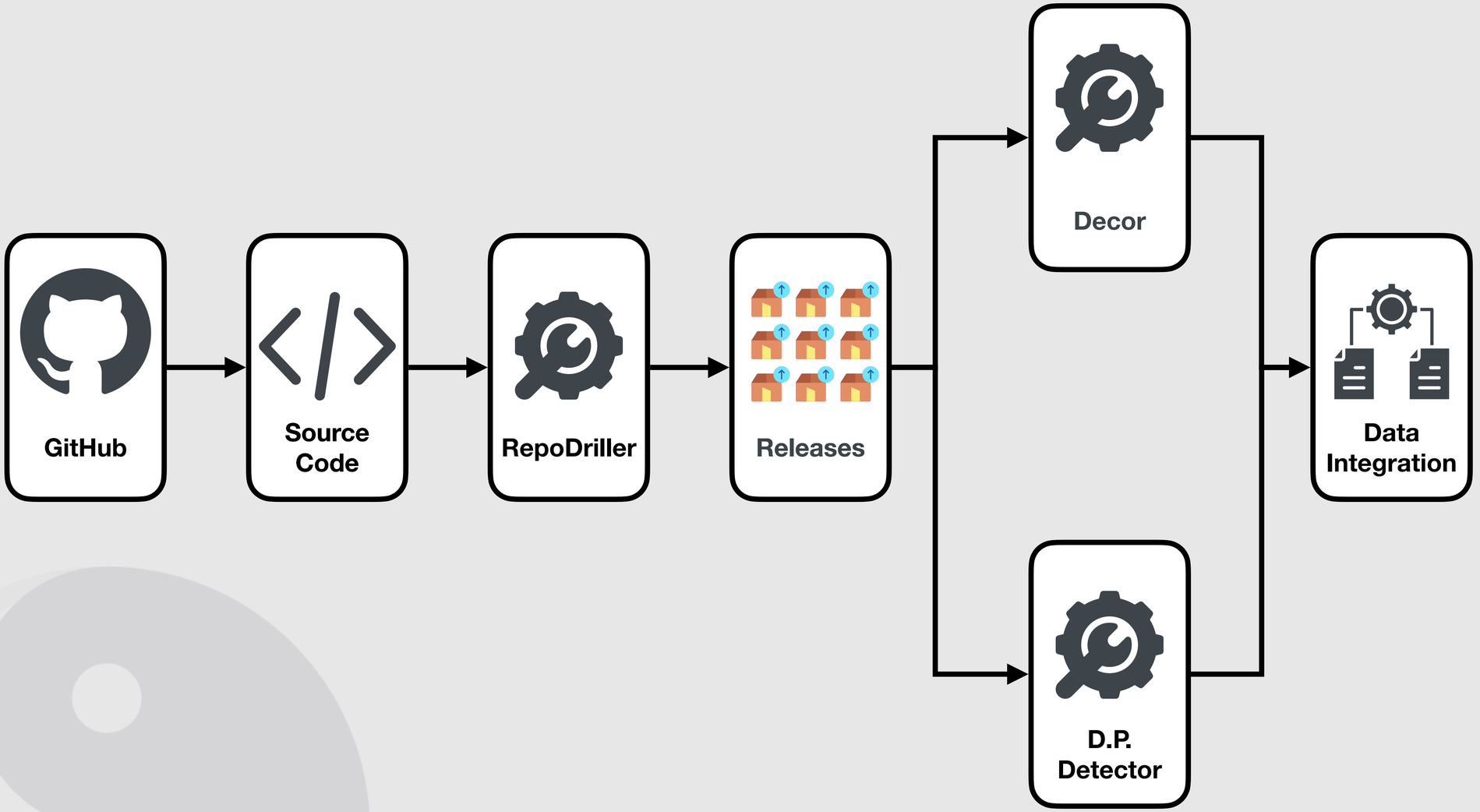
Template Method

Component

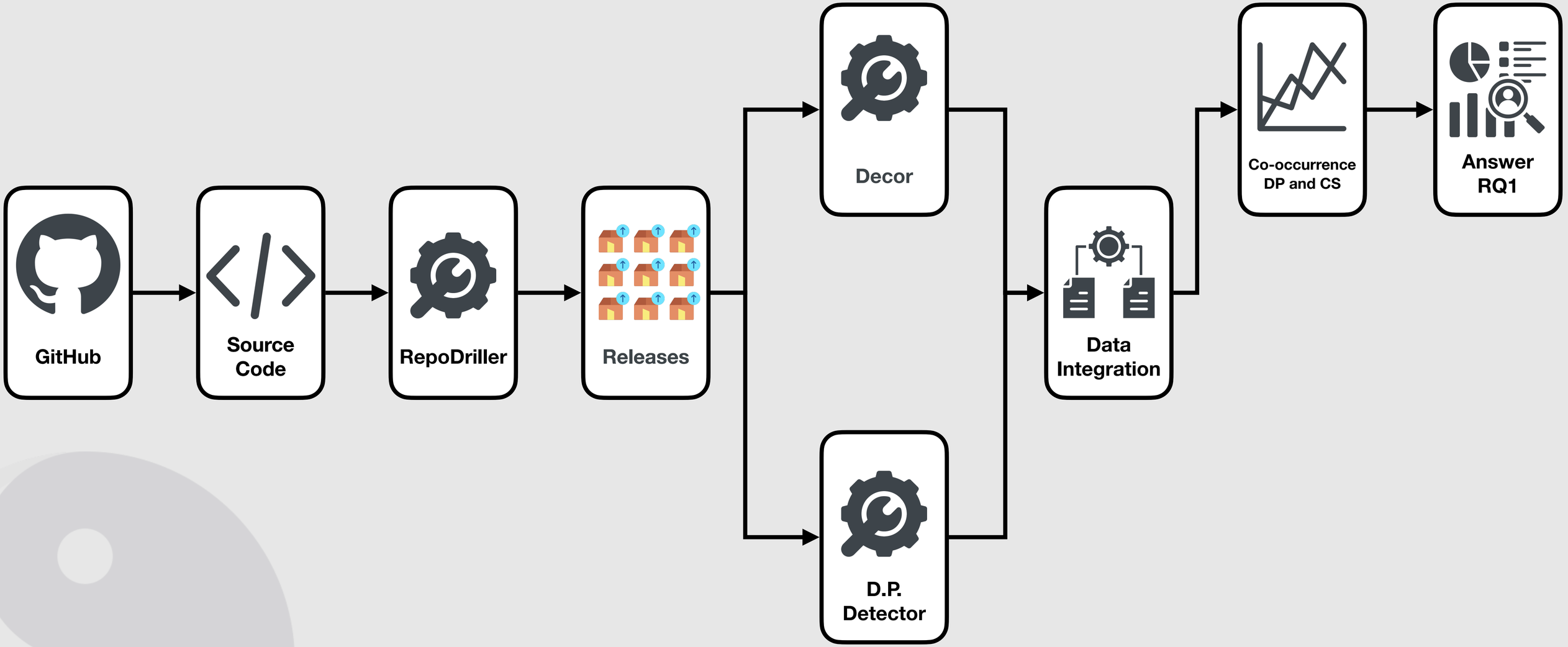
Proxy

Observer

Research Process

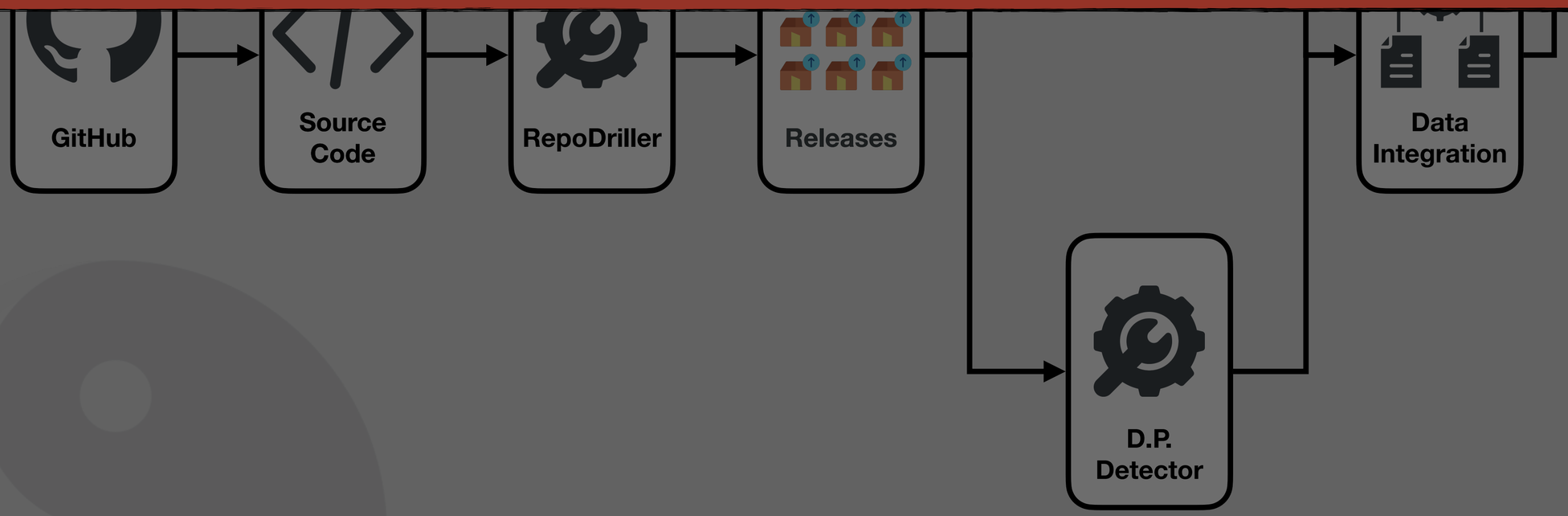


Research Process



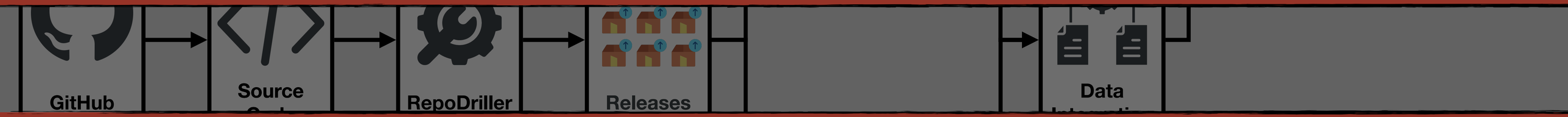
Research Process

We calculate the **frequency** of classes that **participate** to **Design Patterns** and **simultaneity** are **affected** by **Code Smells**



Research Process

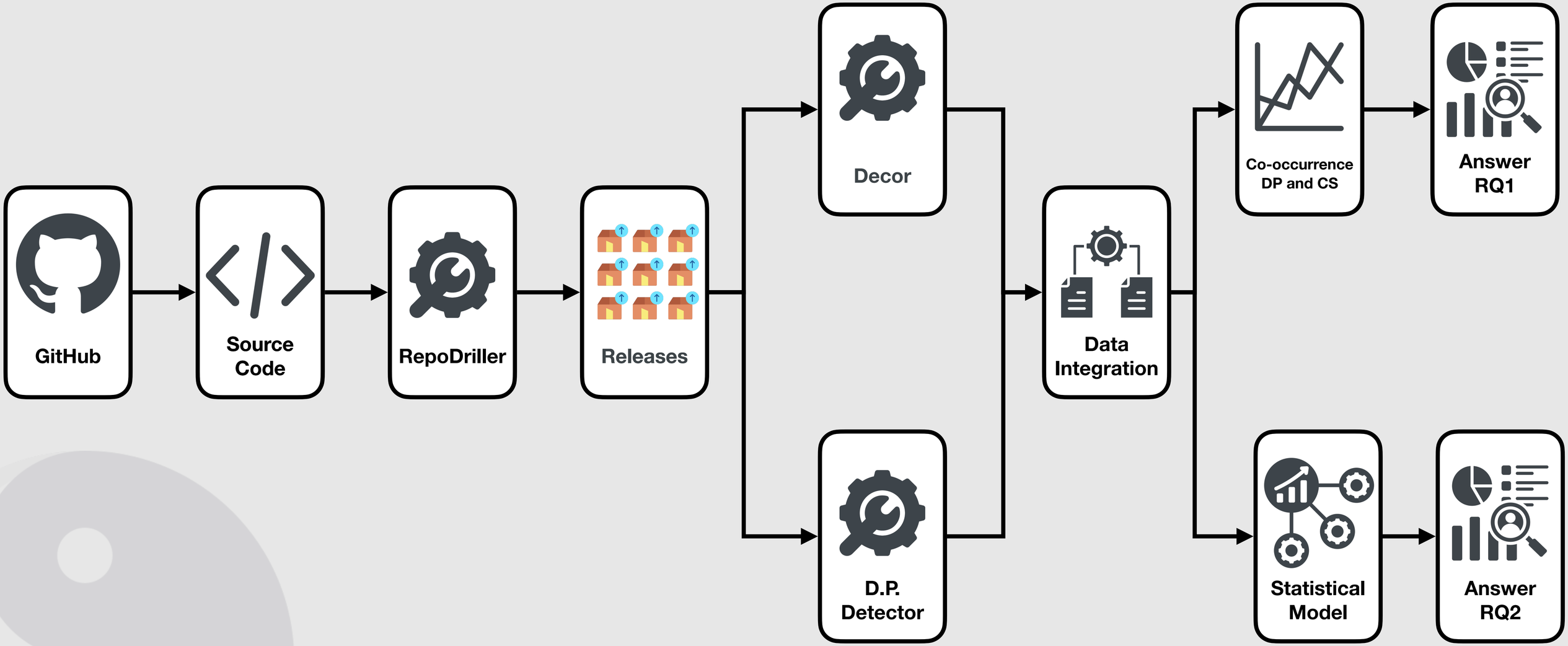
We calculate the **frequency** of classes that **participate** to **Design Patterns** and **simultaneity** are **affected** by **Code Smells**



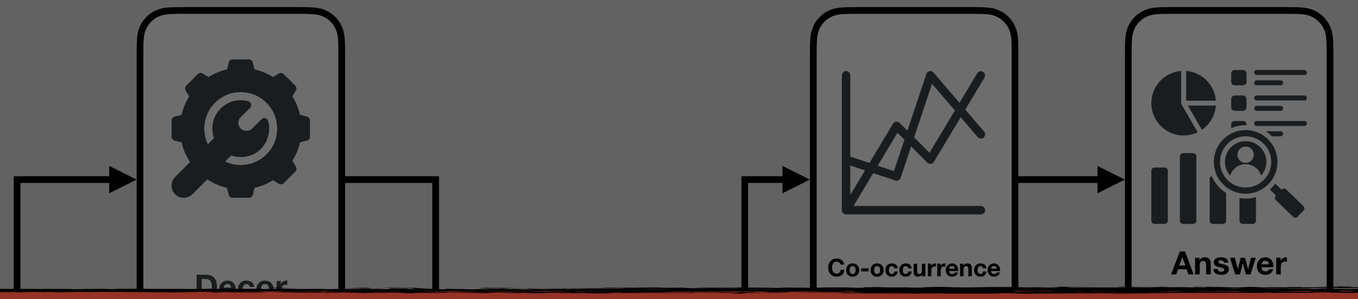
We **normalize** the data using **MIN-MAX** and then **plot** the **results**

Detector

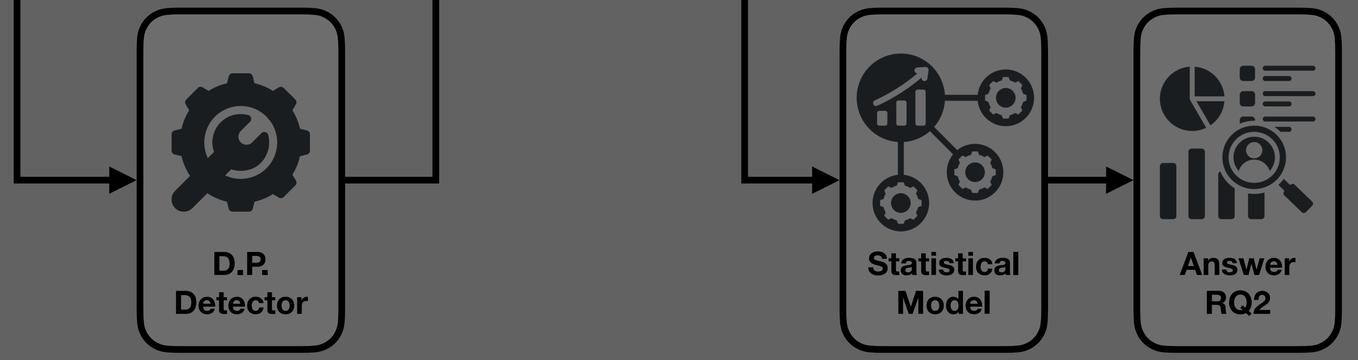
Research Process



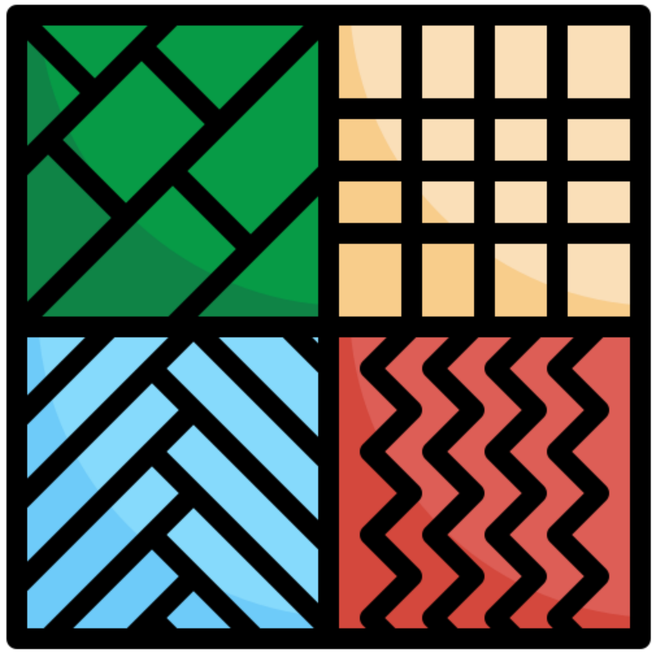
Research Process



To assess the results of the RQ2, we use the
Generalized Linear Model

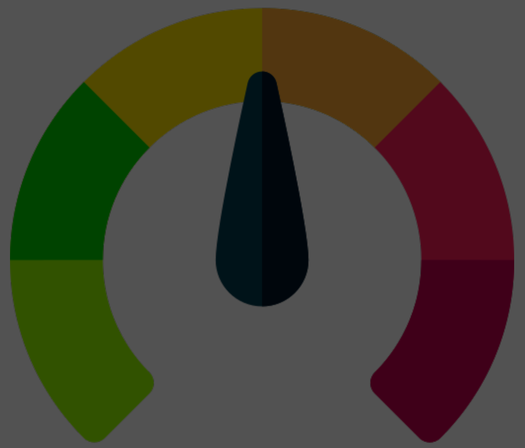


Variables of our model



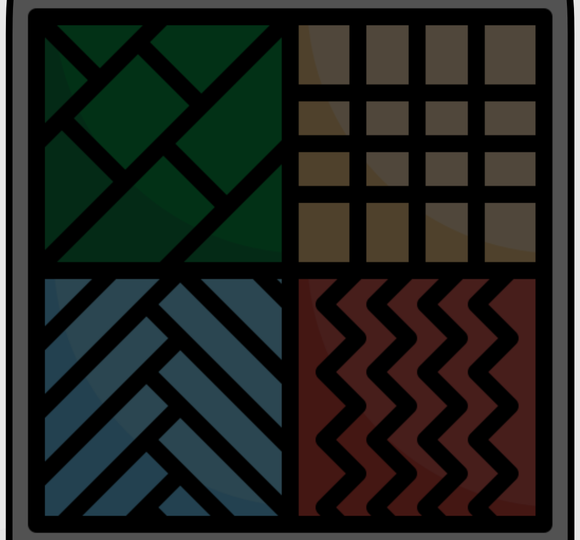
Independent Variable

Dependent



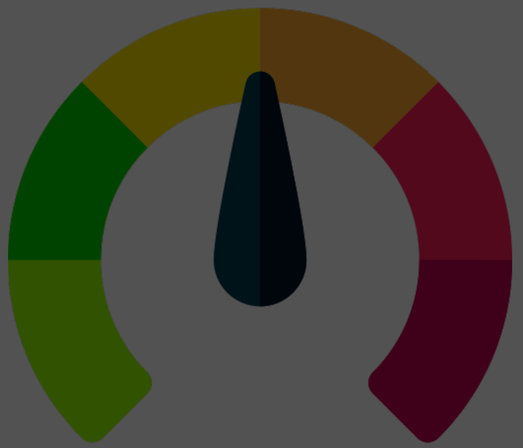
Control

Variables of our model



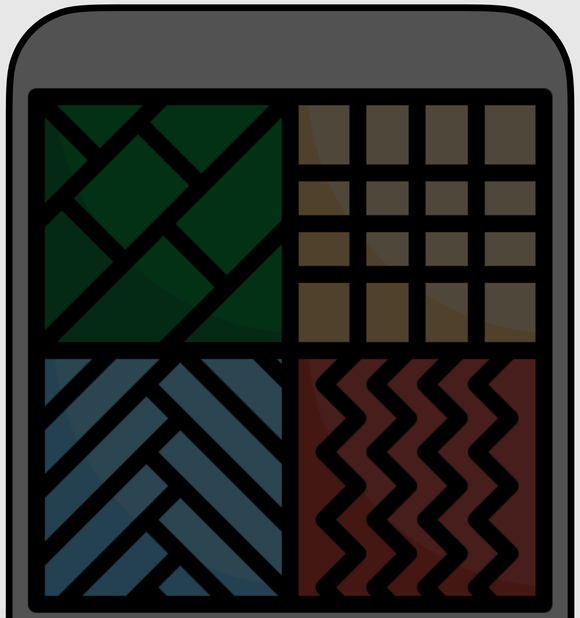
Independent

Dependent Variable



Control

Variables of our model



Independent

Dependent

Control Variable

Variables of our model

A **positive** coefficient indicates a **positive** correlation between the **independent variables and** the **dependent variable**

A **negative** coefficient indicates a **negative** correlation between the **independent variables** and the **dependent variable**

Co-occurrences of Design Patterns and Code Smells

46%

projects contain **no instances** of classes that participate in **Design Patterns** and **simultaneously** are affected by **Code Smells**

46%

projects contain
no instances of classes that
participate in **Design Patterns**
and **simultaneously**
are affected by **Code Smells**

54%

projects contain **instances**
of classes that participate in
Design Patterns
and **simultaneously**
are affected by **Code Smells**

Co-occurrence of Design Patterns and Code Smells

In all projects where exists a **co-occurrence** between **Design Patterns** and **Code Smells**, the classes implementing **State/Strategy** are affected by **God Class**

In **8** projects the Design Pattern **State/Strategy** was also affected by **Spaghetti Code,** while in other 4 projects **Complex Class** was identified

Co-occurrence of Design Patterns and Code Smells

In all projects where exists a **co-occurrence** between **Design Patterns**

Results might be due to **State/Strategy** implementing **several responsibilities**

In **8** projects the Design Pattern **State/Strategy** was also affected by **Spaghetti Code,** while in other 4 projects **Complex Class** was identified

On the presence of Design Patterns and how they affect Code Smells

60%

of projects are characterized by a **statistical correlation** between **Design Patterns** and **Code Smells**

On the presence of Design Patterns and how they affect Code Smells

On Design Patterns and how they affect Code Smells

	God Class
Adapter/Command	+ + +
Bridge	+ + +
Component	+ + +
Singleton	+ + +
Factory Method	
Template Method	
State/Strategy	
Observer	
Proxy	
Decorator	

+ Low Positive Statistical Correlation
++ Medium Positive Statistical Correlation
+++ Strong Positive Statistical Correlation

- Low Negative Statistical Correlation
- - Medium Negative Statistical Correlation
- - - Strong Negative Statistical Correlation

On Design Patterns and how they affect Code Smells

	God Class	Spaghetti Code
Adapter/Command	+ + +	-
Bridge	+ + +	
Component	+ + +	
Singleton	+ + +	- -
Factory Method		+ + +
Template Method		+ + +
State/Strategy		+ + +
Observer		
Proxy		
Decorator		

+ Low Positive Statistical Correlation
++ Medium Positive Statistical Correlation
+++ Strong Positive Statistical Correlation

- Low Negative Statistical Correlation
- - Medium Negative Statistical Correlation
- - - Strong Negative Statistical Correlation

On Design Patterns and how they affect Code Smells

	God Class	Spaghetti Code	Complex Class
Adapter/Command	+ + +	-	
Bridge	+ + +		
Component	+ + +		
Singleton	+ + +	- -	
Factory Method		+ + +	
Template Method		+ + +	
State/Strategy		+ + +	
Observer			
Proxy			
Decorator			

+ Low Positive Statistical Correlation
++ Medium Positive Statistical Correlation
+++ Strong Positive Statistical Correlation

- Low Negative Statistical Correlation
- - Medium Negative Statistical Correlation
- - - Strong Negative Statistical Correlation

On Design Patterns and how they affect Code Smells

	God Class	Spaghetti Code	Complex Class
Adapter/Command	+ + +	-	
Bridge	+ + +		
Template Method	+ + +		
State/Strategy	+ + +		
Observer			
Proxy			
Decorator			

Although the findings of the RQ1 show the **co-occurrences** between several **Design Patterns** and the **Complex Class**, the results indicate that **there is no correlation**

+ Low Positive Statistical Correlation
++ Medium Positive Statistical Correlation
+++ Strong Positive Statistical Correlation

- Low Negative Statistical Correlation
- - Medium Negative Statistical Correlation
- - - Strong Negative Statistical Correlation

The presence of **Design Patterns does not** necessarily **guarantee** a **high quality**, as they might be correlated with **Code Smells**

An **abuse** or **misuse** of **Design Patterns** can lead to an **increase** the **Code Complexity** and a **decrease** of **Code Comprehension**

An **abuse** or **misuse** of **Design Patterns** can lead to an **increase** the **Code Complexity** and a **decrease** of **Code Comprehension**

The introduction of **Design Patterns** should be carefully **planned** at **design time**, to resolve **specific problems**, to **avoid** making **sub-optimal choices**

Summing up

Summing up

We analyzed over 540 releases of
15 Java projects

Summing up

We analyzed over 540 releases of
15 Java projects

Classes participating in Design Patterns are
often affected by Code Smells themselves

Summing up

We analyzed over 540 releases of 15 Java projects

Classes participating in Design Patterns are often affected by Code Smells themselves

Out of 10 Design Patterns analyzed, 7 showed a positive correlation with the presence of at least one Code Smell

Future Work

are
es

wed
f at

Future Work

Extend the dataset

Future Work

Extend the dataset

Understand the developers’ perspective on the impact of Design Pattern on Code Smells

Future Work

Extend the dataset

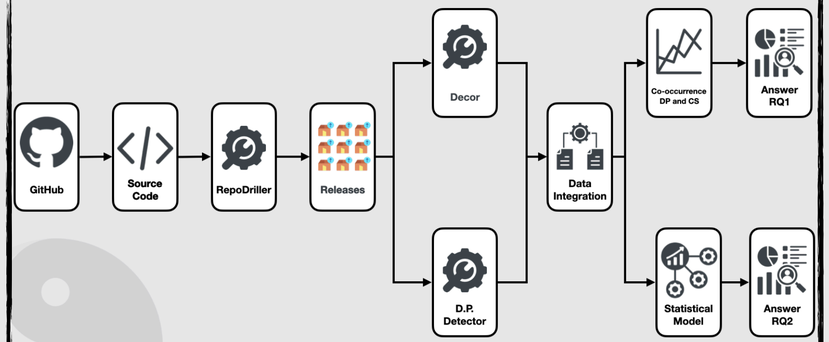
Understand the developers’ perspective on the impact of Design Pattern on Code Smells

Understand how developers implemented Design Patterns to evaluate the correctness

Goal

investigating **whether** and **how** **design patterns** are **related** to the **emergence** of issues compromising code **understandability**

Reseach Process



46%

are **no instances** of classes that participate in **design patterns** and **simultaneously** are affected by **code smells**

54%

are **instances** of classes that participate in **design patterns** and **simultaneously** are affected by **code smells**

60% of projects

We found that the implementation of **design patterns** determined the **presence** of **code smells** in a **statistically significant** way

sesa lab

SOFTWARE ENGINEERING SALERNO

SCAN ME!
I'm the paper

giagiordano@unisa.it
giammariagiordano.github.io/giammaria-giordano
@giammariagiord1