**Noname manuscript No.**
(will be inserted by the editor)

# Investigating Design Anti-pattern and Design Pattern Mutations and Their Change- and Fault-proneness

**Zeinab (Azadeh) Kermansaravi · Md Saidur Rahman · Foutse Khomh · Fehmi Jaafar · Yann-Gaël Guéhéneuc**

Received: date / Accepted: date

**Abstract** During software evolution, inexperienced developers may introduce design anti-patterns when they modify their software systems to fix bugs or to add new functionalities based on changes in requirements. Developers may also use design patterns to promote software quality or as a possible cure for some design anti-patterns. Thus, design patterns and design anti-patterns are introduced, removed, and mutated from one another by developers.

Many studies investigated the evolution of design patterns and design anti-patterns and their impact on software development. However, they investigated design patterns or design anti-patterns in isolation and did not consider their mutations and the impact of these mutations on software quality. Therefore, we report our study of bidirectional mutations between design patterns and design anti-patterns and the impacts of these mutations on software change- and fault-proneness.

We analyzed snapshots of seven Java software systems with diverse sizes, evolution histories, and application domains. We built Markov models to capture the probability of occurrences of the different design patterns and de-

Zeinab Kermansaravi
SWAT Lab, Ptidej Team, DGIGL, Polytechnique Montréal, Montréal, QC, Canada
E-mail: zeinab.kermansaravi@polymtl.ca

Md Saidur Rahman
SWAT Lab, DGIGL, Polytechnique Montréal, Montréal, QC, Canada
E-mail: saidur.rahman@polymtl.ca

Foutse Khomh
SWAT Lab, DGIGL, Polytechnique Montréal, Montréal, QC, Canada
E-mail: foutse.khomh@polymtl.ca

Fehmi Jaafar
Computer Research Institute of Montréal, Montréal, QC, Canada
E-mail: fehmi.jaafar@crim.ca

Yann-Gaël Guéhéneuc
Ptidej Team, CSSE, Concordia University, Montréal, QC, Canada
E-mail: yann-gael.gueheneuc@concordia.ca

sign anti-patterns mutations. Results from our study show that (1) design patterns and design anti-patterns mutate into other design patterns and–or design anti-patterns. They also show that (2) some change types primarily trigger mutations of design patterns and design anti-patterns (renaming and changes to comments, declarations, and operators), and (3) some mutations of design anti-patterns and design patterns are more faulty in specific contexts.

These results provide important insights into the evolution of design patterns and design anti-patterns and its impact on the change- and fault-proneness of software systems.

**Keywords** Design smells · Design patterns · Anti-patterns · Fault-proneness · Change-proneness · Markov Chain

## 1 Introduction

### 1.1 Background and Motivation

Quality assurance is one of the most critical challenges in software development and evolution [1]. Under tight deadlines and other business constraints, developers may take poor design or coding decisions and may follow bad practices. These poor design decisions and bad coding practices are collectively called "smells" [2,3].

Smells are divided into several different categories [4], such as code smells, design smells, lexical smells, etc. Code smells include low-level problems in the source code that may be symptoms of poor coding practices [5,6]. Design smells include design anti-patterns that describe poor solutions to recurring design problems. Design anti-patterns have been reported to make software development and evolution difficult [7]. They affect program comprehension [8] and increase change- and fault-proneness [7].

Opposite to design anti-patterns, design patterns are good solutions to recurring design problems, which promote code reuse and increase reliability, readability, and flexibility [1]. Gamma *et al.* [1] suggested using specific design patterns to ease evolution and increase reuse and flexibility. Studies showed that design patterns often increase the quality of software systems (*e.g.,* [9]). Yet, a few studies showed that some design patterns can decrease some quality attributes [10].

Recent studies [11–13] showed the existence of relationships between design patterns and design anti-patterns. Such relationships can help developers to understand their systems better and simplify development and evolution [11]. Yet, no study considered that design patterns sometimes (d)evolve into design anti-patterns and analysed the impact of such mutations on software quality.

Because design anti-patterns negatively affect software quality while design patterns improve it, understanding the evolution of design patterns and design anti-patterns into one another could help developers identify the riskiest design anti-patterns, avoid introducing design anti-patterns, and–or avoid evolving design patterns into such design anti-patterns.

### 1.2 Research problem

In previous works [11,14], we investigated the static relationships between design anti-patterns and design patterns and how these relationships evolve in time. We studied the relationships between classes playing roles in design patterns and design anti-patterns and reported that static relationships between design patterns and design anti-patterns exist but they are temporary. We also showed that classes playing roles in design anti-patterns and having relationships with design patterns are more change-prone but are less fault-prone than other classes.

In another previous work, [10], we also showed that the design of systems degrades over time, presumably due to the removal (or lack of use) of design patterns and the introduction of design anti-patterns.

Thus, these studies (and others) considered design patterns and design anti-patterns as unique, atomic entities in each releases of the studied systems. Yet, during software evolution, design (anti)patterns *do* evolve, appearing, disappearing, and mutating into one another. Understanding this evolution of design patterns and design anti-patterns across releases, and in particular their mutations into one another, could help developers avoid mutations that negatively impact software quality while promoting those that improve it.

### 1.3 Contributions

This study is a quasi-replication of a previous study by Jaafar *et al.* [14]. Some of the research questions in this paper are similar to those in the previous study [14], which pertain to:

- Computing the probability of design anti-patterns mutations using Markov models.
- Comparing classes with and without design anti-patterns and their fault-proneness.

Moreover, in this study, we consider both design patterns and design anti-pattern mutations during software evolution. We examine the impacts of design patterns and design anti-patterns mutations on change- and fault-proneness. We investigate seven open-source Java software systems to answer to the the following five research contributions:

1. We study how design patterns and design anti-patterns mutate over time using Markov models.
2. We study the types of changes that occur during design anti-patterns mutations.
3. We study the impact of these mutations on fault-proneness.
4. We study the types of changes that lead to design patterns and design anti-patterns mutations.
5. We study the most fault-prone transitions between design patterns and design anti-patterns.

We use seven different open-source systems of different sizes and from different application domains: Apache Ignite[1], Apache Solr [2], Eclipse IDE [3], Matsim [4], Mule [5], Nuxeo [6], and Ovirt [7].

We consider thirteen design anti-patterns from [15] and eight design patterns [1]. For the detection of design anti-patterns and design patterns, we use DECOR [16] and DeMIMA [17]. We first detect the occurrences of design anti-patterns and design patterns in all the studied releases of the systems and then investigate the types of mutations: persistent, deleted, introduced, and changed between these snapshots.

Second, we build Markov models [18] to compute the probability values of such mutations. We build one Markov model per studied system. In the models, design anti-patterns and design patterns are nodes while the probabilities of their mutations into one another label the edges between nodes. We compute the probability values by analyzing all the releases of the software systems during the considered period of time.

Third, we use the SZZ algorithm [19] to find fault-inducing commits and investigate the impact of design patterns and–or design anti-patterns mutations on the fault-proneness of classes.

Fourth, we define thirteen types of change and study them to discover the kinds of changes leading to mutations between design patterns and design anti-patterns. We also study the effects of the change types on fault-proneness.

### 1.4 Research Questions

We use the seven open-source Java software systems to answer the following four research questions:

- **RQ1:** *Do design patterns and–or design anti-patterns mutate during software evolution and what is the probability of the mutations?* We build Markov models [18] showing which design patterns and–or design anti-patterns mutate into one another during a studied period of evolution. We consider both appearance and disappearance of design patterns and design anti-patterns. We observe that both design patterns and design anti-patterns mutate in the systems. We compute the probabilities of all possible mutations using the Markov models. We also present the most frequent mutations of design patterns and design anti-patterns along the following four mutation types:
  1. Design anti-patterns to some other design anti-patterns;

[1] `https://ignite.apache.org/`
[2] `http://lucene.apache.org/solr/`
[3] `https://www.eclipse.org/`
[4] `https://matsim.org/`
[5] `http://www.mulesoft.org/`
[6] `https://www.nuxeo.com/`
[7] `https://www.ovirt.org/`

2. Design anti-patterns to design patterns;
3. Design patterns to design anti-patterns;
4. Design patterns to some other design patterns.

- **RQ2:** *What types of changes lead to mutations between design patterns and design anti-patterns?* Design patterns and design anti-patterns evolve through different types of changes as the system evolves. We define thirteen change types and investigate classes experiencing these change types and participating in design patterns and–or design anti-patterns. We see that different types of changes occur during the evolution of software systems and lead to different mutations. We study the impact of the types of changes on mutations between design patterns and–or design anti-patterns. We present the most prevalent type of changes leading to mutations.
- **RQ3:** *What is the fault-proneness of mutated design patterns and anti-patterns and what transitions lead to more fault-prone mutations?* Design patterns and design anti-patterns may frequently mutate in other types of patterns during the evolution process. We study whether such mutations are risky regarding fault-proneness. We also present the riskiest transitions among design patterns and design anti-patterns. We observe that classes participating in mutated design anti-patterns are more fault-prone than classes involved in mutated design patterns. We also see that mutations between design anti-patterns and design patterns are more faulty than other mutations.
- **RQ4:** *Do specific types of changes lead to increased fault-proneness during mutations?* We investigate whether the types of changes to design patterns and design anti-patterns impact fault-proneness. We examine faulty-classes and check whether a mutation occurred during the evolution of these classes. We also examine all changes experienced by the classes during the evolution of the systems. We observe that some of the change types make the systems more fault-prone. We study whether specific types of changes that cause the mutations between design patterns and design anti-patterns are more fault-prone.

The results of these four research questions show that there is a high probability for some design patterns and design anti-patterns to mutate to others types of design patterns and design anti-patterns. The changes that lead to the mutations are mostly structural changes, in particular the addition of large number of attributes or long methods. Results also show that some mutations increase the fault-proneness of the analysed software systems.

### 1.5 Organization

The rest of the paper is organized as follows: Section 2 describes the related work. Section 3 presents the methodology of our study. Section 4 reports its experimental setup. Section 5 presents its results. Section 6 discusses the results and Section 7 threats to their validity. Finally, Section 8 concludes the paper with future work.

## 2 Related Work

### 2.1 Design Anti-pattern and Detection

Webster *et al.* [20] describes design anti-pattern as a solution to a problem that is used frequently but negatively affects software quality. Riel *et al.* [21] proposed 61 heuristics of good object-oriented programming that can be used to manually assess a program quality for improving its design and implementation. These heuristics are similar to code smells. Later, Brown *et al.* [15] introduced 40 types of design anti-patterns that form the basis of design anti-patterns detection approaches [5,22,16,23].

Several approaches have been proposed to detect design anti-patterns. Van Emden *et al.* [5] developed JCosmo to visualize the code layout and locate anti-patterns. JCosmo uses primitives and rules to detect design anti-patterns while parsing the source code into an abstract model (similar to the Famix meta-model [24]). The goal is to evaluate code quality and help developers to do refactoring. Marinescu *et al.* [25] combined detection strategies and additional information collected from the documentation of problematic structures in the histories of software systems to improve the detection results.

Settas *et al.* [23] proposed Bayesian network-based approach to improve the detection of design anti-patterns. Their approach leverages probabilistic knowledge, which contains the relationships of design anti-patterns regarding their causes, symptoms, and consequences. iPlasma [22] detects design anti-patterns by calculating metrics on C++ or Java source code and by applying some rules that combine the metrics.

In this paper, we use DECOR to specify and detect design anti-patterns because of its higher detection accuracy [16] and wider domain coverage. We present a detailed description of DECOR in Section 3.1.

### 2.2 Design Pattern and Detection

Design patterns in object-oriented software development and their detection have been well-studied in the past two decades [1,26]. Kramer *et al.* [26] introduced an approach to detect design information directly from C++ header files. Design patterns are represented as Prolog rules that query this design information. Their approach detects five structural design patterns: Adapter, Bridge, Composite, Decorator, and Proxy.

Vokač *et al.* [12] proposed an approach scoring the similarity between the graph of a design pattern and the graph of a system to identify classes participating in this design pattern. Iacob *et al.* [27] identified proven solutions for recurring design problems using design workshops and system analysis. During design workshops, teams of 3-5 developers designed systems while considering design issues. Then, they analysed a set of systems to recognize how developers should consider design problems in the implementation of existing solutions.

In this paper, we use DeMIMA to specify and detect design patterns because of its higher detection accuracy [17], which is described in Section 3.2.

### 2.3 Design Anti-pattern and Design Pattern Evolution and their Impact

Several studies investigated both design anti-pattern and design-pattern evolution. Bieman *et al.* [28] claimed that there is a relative stability in classes participating in design patterns compared to other classes. They showed that large classes are more change-prone than other classes. Vokač *et al.* [12] discussed how different design patterns have different impact on fault-proneness. They studied a large C++ industrial system to prove their claim. Gatrell *et al.* [29] demonstrated that pattern-based classes are more change-prone than other classes. Olbrich *et al.* [30] focused on the historical data of Lucene and Xerces over several years and showed that Blob classes and classes subject to Shotgun Surgery are more change-prone than other classes.

Khomh *et al.* [7] investigated the impact of code smells on the change-proneness of classes. They also studied the influence of design anti-patterns on change- and fault-proneness. They considered 13 design anti-patterns in 54 releases of ArgoUML, Eclipse, Mylyn, and Rhino, and analyzed the probability of classes changing to fix a fault. Their results showed that classes participating in design anti-patterns were significantly more likely to be changed than other classes. This study also investigated two types of changes experienced by classes with design anti-patterns: structural and non-structural changes. Structural changes can modify the class interface while non-structural ones change method bodies. They concluded that structural changes were more likely to occur in classes participating in design anti-patterns.

Yamashita and Moonen [31] reported that developers cannot fully evaluate the overall maintainability of a software system with code smells alone. They argued that different approaches should be combined to achieve complete and accurate evaluations of software maintainability. Taba *et al.* [32] claimed that information about design anti-patterns improves the accuracy of fault prediction models. Jaafar *et al.* [11] empirically studied the relationships between design anti-patterns and design patterns. They showed that some design anti-patterns have relationships with design patterns while others do not.

## 3 Methodology

We now describe the general methodology of our study, shown in Figure 1, which includes the main steps of design (anti-)pattern detection, classification of change types, building Markov models, and detection of faulty classes.

We first extract the source code of the studied systems in snapshots taken every 500 commits in their Git repositories. Then, we detect design anti-patterns and design patterns in all the snapshots of the systems. We then create Markov models based on the detected design anti-patterns and design

patterns to analyze their behaviors during evolution. We also identify change types and faulty classes throughout the period of evolution that we analyzed. We study all the changed types that lead to fault(s) during evolution. We explain each step in details in the following sub-sections.

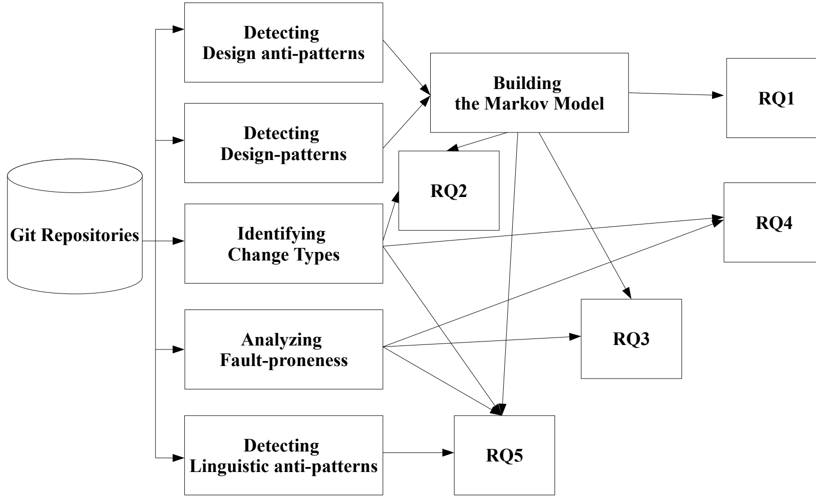


**Fig. 1** Schematic diagram of the methodological steps of the study

### 3.1 Detecting Anti-patterns

We use Defect DEtection for CORection Approach (DECOR) [16] to detect occurrences of design anti-patterns. DECOR offers a domain-specific language to automatically generate design-defect detection algorithms, including design anti-patterns. DECOR uses the Patterns and Abstract-level Description Language meta-model (PADL) [17] and the Primitive, Operators, and Metrics framework (POM) [33] to detect design anti-patterns in object-oriented systems. The output of DECOR is a list of classes and their roles (if any) in occurrences of design anti-patterns. Moha *et al.* [16] reported that DECOR achieves 100% recall while having 31% precision rate in the worst case with an average greater than 60%.

A domain-specific language is more flexible than ad hoc algorithms [16] because domain experts (*i.e.,* developers) can modify the detection rules manually using high-level abstractions, considering the contexts, environments, and characteristics of the analyzed systems. PADL [17] is a meta-model to describe object-oriented systems at different abstraction levels while POM [33] is a PADL-based framework that implements more than 60 metrics.

### 3.2 Detecting Design patterns

We use the Design Motif Identification Multilayered Approach (DeMIMA) [17] to detect occurrences of design patterns. DeMIMA traces design motifs (the micro-architecture describing the solutions of the design patterns) in the source code. It discovers idioms relevant to binary class relationships and then provide an idiomatic model of the source code. The model helps to identify design motifs to create a design model of the system. DeMIMA can recover idioms related to both the relationships among classes and design motifs.

DeMIMA uses explanation-based constraint programming to identify occurrences of design motifs using the roles and relationships describing the motifs in PADL models of systems. It reports the micro-architectures that are occurrences of the motifs, including approximations of the motifs. The output of DeMIMA is a list of classes and their roles (if any) in the occurrences of design patterns. Guéhéneuc and Antoniol [17] report that DeMIMA achieves 100% recall and 34% precision.

### 3.3 Building Mutation Model

We build a Markov model [18] for each studied system to show the mutations between design anti-patterns and design patterns during evolution. For each system, we consider all the patterns whose occurrences we found in its snapshots as nodes in its Markov model. A Markov model shows the set of all possible mutations for one pattern during the evolution of the system.

First, we obtain two files from two consecutive snapshots of a system, $C_0$ and $C_1$, which contain all the occurrences of design patterns and design anti-patterns in each class of each snapshot. Then, the Markov transition matrix of these two files is a square matrix describing the probabilities of one design anti-pattern and–or design pattern mutating into another. Each row contain the probabilities of mutating from one pattern to all the others. Second, we compare the next two snapshots, $C_1$ and $C_2$, and repeat this algorithm up to snapshots $C_{n-1}$ and $C_n$. Then, we sum up all the mutation probabilities for one given pattern into all the others. We report the averaged summed mutation probabilities divided by the total number of each row. The sum of all the probabilities from any pattern to the others is equal to one.

For an example, Figure 2 shows a Markov model whose nodes are design anti-patterns and design patterns and edges represent mutations from one pattern to another. Edges are labeled with the probabilities of the mutation from the source patterns to its mutated patterns. This Markov model shows the mutations of the Builder design pattern across the snapshots of Matsim.

### 3.4 Analyzing Fault-proneness

We use the SZZ algorithm [19] to identify commits that introduce faults in the systems, *i.e.,* fault-inducing commits, and thus faulty classes in the system

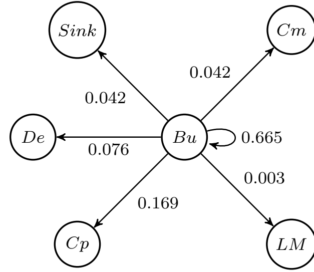


**Fig. 2** Builder (Bu) mutation among the different revisions of Matsim.

snapshots. For each system, we first apply heuristics [34] to link commits to faults. We use regular expressions to detect fault-IDs in the commit logs. Developers use different conventions for these IDs in their systems so, to ensure accuracy, we tune our heuristics on our dataset incrementally and manually.

Given a fault $F$ in a system, we extract from its history the files that fixed the fault (fault-fixing files) using `git diff`. Then, we retrieve the modified and deleted lines from the fault-fixing files. The SZZ algorithm assumes that prior commits that modified these lines are fault-inducing commits.

To identify such prior commits, for each fault-fixing files, we apply `git blame` to retrieve a list of previous commits that modified these files. We filter commits whose submission date is later than the fault creation date. We consider the remaining commits as inducing the fault $B$.

For any $F$, the SZZ algorithm returns a list of commit IDs and fault-inducing files pertaining to $F$. We then use regular expressions to map the object-oriented classes composing the systems with the fault-inducing files.

## 3.5 Identifying Change Types

Different types of changes can affect software systems with different impacts on fault-proneness. For example, changes to comments are less likely to lead to faults than changes to method invocations. Table 1 shows the change types that we consider, which were also considered in a previous work [35].

We use srcML [36] to transform each file in the snapshots of a system into an XML document, in which code elements are tagged by type or function, *e.g.,* a class declaration, a parameter list, or a control flow statement. Then, we compare the srcML tags between each two subsequent snapshots and extract their differences. The removed tags from the older snapshot and the added tags in the newer snapshot are *changed tags*. We manually group the unique changed tags into a series of *change types*. Table 1 shows the change types and their corresponding srcML (changed) tags. We group some change types together, which have similar impacts on source code. Changes in a same group are likely to have similar impacts on fault-proneness.

**Table 1** Change types identified from the source code of the systems studied

| **Change type** | **srcML tag(s)** |
|---|---|
| Access | *super, public, private, protected, extern* |
| Class | *extends, class, interface, implements, class_decl* |
| Code block | *expr_stmt, expr, block* |
| Comment | *annonation, comment, @type, @format* |
| Control flow | *while, do, if, else, elseif, break, goto, for, foreach, control, continue, then, switch, case, return, incr, default, condition* |
| Declaration | *decl_stmt, modifier, specifier, decl, function_decl, literal, label, empty_stmt, construction_decl, annonation_dfn* |
| Exception | *assert, try, catch, throw, throws, finally* |
| Import | *import, package* |
| Invocation | *call* |
| Method | *constructor, default, static, type, lambda, function, function_decl, unit* |
| Operator | *index, synchronized, enum, operator, ternary* |
| Parameter | *argument, param, parameter_list, argument_list, parameter* |
| Renaming | *renaming, name* |

For a given file $F$, in a specific snapshot $S$, our approach yields a list of change types listed in Table 1. Because we study design (anti-)patterns from commits; in each selected snapshots, we aggregate the change types related to $F$ in the commits $\{C_1, C_2, ..., C_n\}$, which form that snapshot.

## 4 Experimental Setup

We now describe the systems and patterns making our experimental setup.

### 4.1 Subject Systems

We consider seven Java-based open-source systems for our study, summarised in Table 2. We select these systems based on diversity in code size, application domains, and evolution histories. The number of lines of code of these systems range from hundred of thousand to several millions. They belong to different domains, from IDE to database. Some were used in previous studies, which allows some comparisons. These systems have evolved over the years and have many commits/versions to provide a dataset for analyzing pattern mutations and fault-proneness.

However, the choice of systems is inherently a threat to the conclusion and generalisability of any empirical study, which we acknowledge in Section 7.

We now briefly describe the subject systems.

**Eclipse IDE for Java** is an IDE for Java developers. The IDE offers the Java Development Tools (JDT) to develop Java systems. It contains also CVS, SVN, and Git clients. It also includes an XML editor, Mylyn as a task management system, build supports for Maven and WindowBuilder, etc.

**Table 2** Analyzed systems

| System | Applicaion domain | # Commits | LOC | Issue Tracker |
|---|---|---|---|---|
| Eclipse for Java | IDE | 281,396 | 9,064,794 | Bugzilla |
| Nuxeo Platform | Colaboration management | 265,380 | 5,741,131 | Jira |
| oVirt | Visualization platform | 149,128 | 2,764,655 | Bugzilla |
| Matsim | Transportation management | 44,200 | 1,602,877 | Atlassian |
| Apache Solr | Search server | 30,995 | 658,711 | Jira |
| Apache Ignite | Distributed DB platform | 24,104 | 1,471,036 | Jira |
| Mule Community Edition | Integration platform | 22,891 | 309,616 | Jira |

**Nuxeo**, also called Nuxeo Platform, is an open-source context management and collaboration platform, which provides different information management solutions for developers to build business applications.
**oVirt** is a visualization management platform in Java. It provides a centralized management of resources, storage, and virtual machines, which allows managing enterprise infrastructure.
**Matism** is a framework to build large-scale transport simulations. Its development team provides a comprehensive documentation for users and developers to ease usability and maintainability.
**Apache Solr** from the Apache Lucene project is an open-source Java search server for Web sites, databases, and files. It is popular and fast, using Lucene Java search library at its core. It runs as a standalone full-text search server.
**Apache Ignite** is a in-memory computing platform used as database and caching system. It helps solving problems related to speed and scalability and can be used to speed up relational and NoSQL databases.
**Mule** is the run-time engine of a Java-based enterprise service bus (ESB) and integration platform. Developers can connect applications quickly and easily to exchange data. It allows service creation and hosting, service mediation, message routing, and data transformation.

## 4.2 Analyzed Patterns

### *4.2.1 Anti-patterns*

We select thirteen anti-patterns in our study. These anti-patterns introduced by Brown *et al.* [15] express problems with data, complexity, size, and the features related to classes [7]. They have been studied in previous work [7]. We summarize their definitions below, details are available elsewhere [37]:

- AntiSingleton (AS): A class that provides mutable class variables, which could be used as global variables.
- Blob (Bl) or God Class (GC): A class that is too large and not cohesive enough, which monopolizes most of the system processing, takes most of the decisions, and is associated to data classes.
- ClassDataShouldBePrivate (CS): A class that exposes its fields, thus violating the principle of encapsulation.

- ComplexClass (CC): A class that has (at least) one large method and complex method, in terms of cyclomatic complexity and line of codes LOCs.
- LargeClass (LC): A class that has (at least) one large method, in terms of LOCs.
- LazyClass (LZC): A class that has few fields and methods that are complex.
- LongMethod (LM): A class that has (at least) one method that is overly long, in terms of LOCs.
- LongParameterList (LP): A class that has (at least) one method with a long list of parameters with respect to the average numbers of parameters per methods.
- MessageChain (MCh): A class that uses a long chain of method invocations to realize one of its functionality.
- RefusedParentBequest (RP): A class that overrides methods using empty bodies.
- SpaghettiCode (SC): A class declaring long methods which do not have any parameters. These methods are complex, with a complicated control flow. The class does not use polymorphism and–or inheritance.
- SpeculativeGenerality (SG): A class that is defined as abstract but that has very few children, which do not make use of its methods.
- SwissArmyKnife (SA): A class whose methods can be divided into disjoint sets, providing different, unrelated functionalities.

#### *4.2.2 Design Patterns*

We consider eight design patterns presented in Gamma *et al.* [1], which we select due to their popularity and because previous works also studied them [38,39]. Their complete definitions and specifications are available in [39,40]:

- Builder (Bu): A pattern to separate the construction of a complex object from its representation.
- Command (Cm): A pattern to encapsulate a request as an object.
- Composite (Cp): A pattern that composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Decorator (De): A pattern that attaches additional responsibilities to an object dynamically. Decorator provides a flexible alternative to sub-classing for extending functionality.
- Factory Method (FM): A pattern that defines an API for object creation in which subclasses choose the class to instantiate.
- Observer (Ob): A pattern that defines a one-to-many dependency between objects to notify all the object dependent on one object when it changes.
- Prototype (Pt): A pattern that specifies the kind of objects to create using a prototypical instance.
- Singleton (Si): A pattern that restricts the instantiation of a class to one object to coordinate actions across a system.

## 5 Results and Analysis

We now present the results of our study and answer our five research questions.

### 5.1 **RQ1:** *Do design patterns and–or design anti-patterns mutate during software evolution and what is the probability of the mutations?*

**Motivation:** Understanding the evolution of patterns is important because it can help developers to identify and circumvent risky design patterns and prevent the appearance of design anti-patterns [14]. While some tools can find software entities and their evolution patterns automatically, *e.g.,* [5, 41–43], no previous work investigated the mutation of design (anti-)patterns.
**Computing probability values for all possible mutations:** We apply the detection tools described in Section 3 on snapshots of each of the systems listed in Table 2. Each snapshot contains a large number of classes, which may participate in different types of design anti-patterns and–or design patterns.

We take snapshots every 500 commits in the evolution histories of the systems. This commit interval period is adequate to detect changes occurring between two subsequent snapshots [44, 45].

We automatically compare each two subsequent snapshots to compute the numbers of added or deleted occurrences of design patterns and design anti-patterns. We build one Markov model for each system to show the probabilities of mutations between design patterns and design anti-patterns.

**Table 3** Change probabilities of design anti-patterns and design patterns in Eclipse IDE

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.013 | **0.972** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.015 |
| Bl | 0.007 | **0.27** | **0.447** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.277** |
| CS | 0.001 | 0.000 | 0.001 | **0.993** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 |
| CC | 0.000 | 0.000 | 0.000 | 0.012 | **0.973** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.014 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | **0.5** | **0.5** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.012 | **0.982** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.006 |
| LM | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.024 | **0.95** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.025 |
| LP | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.009 | **0.966** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.025 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.273** | **0.455** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.273** |
| SC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.876** | 0.000 | 0.000 | 0.000 | 0.000 | 0.049 | 0.000 | 0.002 | 0.05 |
| Cm | 0.000 | 0.010 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.668** | 0.091 | 0.09 | 0.000 | 0.000 | 0.000 | 0.000 | **0.141** |
| Cp | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.042 | **0.169** | **0.665** | 0.076 | 0.000 | 0.000 | 0.000 | 0.042 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.892** | **0.107** | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.020 | **0.966** | 0.014 |

**Table 4** Change probabilities of design anti-patterns and design patterns in Nuxeo

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.014 | **0.969** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.017 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bl | 0.003 | **0.283** | **0.417** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.297** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CS | 0.002 | 0.000 | 0.008 | **0.982** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CC | 0.001 | 0.000 | 0.000 | 0.040 | **0.912** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.048 | **0.910** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.042 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LM | 0.001 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.028 | **0.937** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.030 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LP | 0.000 | 0.001 | 0.000 | 0.006 | 0.000 | 0.000 | 0.002 | 0.017 | **0.946** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.028 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SC | 0.012 | 0.000 | 0.000 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.067 | **0.824** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.085 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cm | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.019 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.133** | 0.000 | 0.000 | **0.105** | **0.760** | 0.000 |

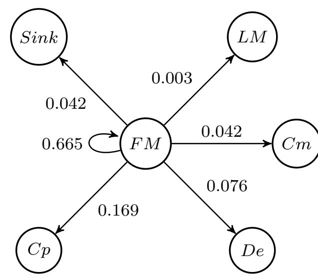


**Fig. 3** FactoryMethod (FM) mutation in Eclipse.

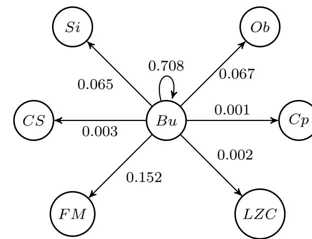


**Fig. 4** Builder (Bu) mutation in Nuxeo.

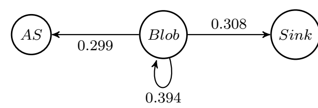


**Fig. 5** Blob (Bl) mutation in oVirt.

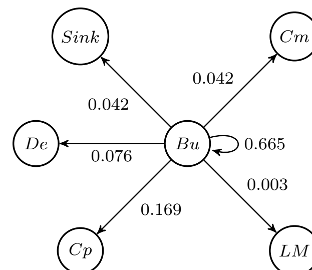


**Fig. 6** Builder (Bu) mutation among the different snapshots of Matsim.

**Table 5** Change probabilities of design anti-patterns and design patterns in oVirt

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.018 | **0.971** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.011 |
| Bl | 0.000 | **0.299** | **0.394** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.308** |
| CS | 0.006 | 0.000 | 0.003 | **0.982** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.009 |
| CC | 0.000 | 0.000 | 0.000 | 0.012 | **0.975** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.013 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | **0.998** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 |
| LM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.012 | **0.977** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.011 |
| LP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | **0.969** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.022 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.082 | **0.857** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.061 |
| SC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cm | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.019 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.097** | **0.798** | **0.103** |

**Table 6** Change probabilities of design anti-patterns and design patterns in Matsim

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.066 | **0.893** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.040 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bl | 0.003 | **0.372** | **0.279** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.346** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CS | 0.025 | 0.000 | 0.037 | **0.9** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CC | 0.004 | 0.001 | 0.002 | 0.075 | **0.848** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.069 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | **0.1** | **0.831** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.066 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LM | 0.003 | 0.001 | 0.002 | 0.013 | 0.000 | 0.000 | 0.063 | **0.86** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.059 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LP | 0.003 | 0.000 | 0.003 | 0.013 | 0.000 | 0.000 | 0.007 | 0.034 | **0.887** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.053 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.021 | **0.596** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.193** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SC | 0.026 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.038 | **0.834** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.099** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.091 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.182** | **0.455** | 0.000 | 0.000 | 0.000 | 0.000 | **0.273** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.546** | 0.000 | 0.001 | **0.272** | 0.000 | **0.141** | 0.000 | 0.030 | 0.000 |
| Cm | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.030 | 0.000 | **0.371** | 0.075 | **0.387** | **0.117** | 0.012 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.037 | 0.044 | **0.850** | 0.061 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.001 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.036 | 0.074 | **0.825** | 0.059 | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.049 | **0.912** | 0.034 | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.028 | 0.000 | 0.000 | 0.009 | **0.962** | 0.000 |

**Table 7** Change probabilities of design anti-patterns and design patterns in ApacheSolr

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bl | 0.000 | **0.321** | **0.365** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.313** |
| CS | 0.000 | 0.000 | 0.009 | **0.983** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 |
| CC | 0.000 | 0.000 | 0.000 | 0.033 | **0.93** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.037 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.072 | **0.859** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.069 |
| LM | 0.000 | 0.000 | 0.001 | 0.002 | 0.000 | 0.000 | 0.033 | **0.928** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.036 |
| LP | 0.000 | 0.001 | 0.000 | 0.008 | 0.000 | 0.000 | 0.007 | 0.079 | **0.779** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.125 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.088 | **0.84** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.072 |
| SC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.013 | 0.000 | **0.981** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.006 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.963** | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 | 0.000 | 0.000 | 0.027 |
| Cm | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.012 | **0.874** | 0.005 | 0.000 | 0.000 | 0.000 | 0.092 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.996** | 0.000 | 0.000 | 0.000 | 0.004 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.007 | **0.981** | 0.012 |

**Table 8** Change probabilities of design anti-patterns and design patterns in ApacheIgnite

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bl | 0.000 | **0.375** | **0.33** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.295** |
| CS | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CC | 0.000 | 0.000 | 0.000 | 0.053 | **0.905** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.042 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.015 | **0.97** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.015 |
| LM | 0.000 | 0.000 | 0.000 | 0.004 | 0.000 | 0.000 | 0.045 | **0.905** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.047 |
| LP | 0.000 | 0.000 | 0.000 | 0.009 | 0.000 | 0.000 | 0.006 | 0.051 | **0.851** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.081 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.976** | 0.000 | 0.000 | 0.000 | 0.000 | 0.004 | 0.000 | 0.000 | 0.018 |
| Cm | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | **0.994** | 0.003 |

**Table 9** Change probabilities of design anti-patterns and design patterns in Mule

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.032 | **0.937** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.030 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bl | 0.000 | **0.313** | **0.313** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.374** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CS | 0.003 | 0.000 | 0.006 | **0.963** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.028 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| CC | 0.001 | 0.000 | 0.000 | 0.071 | **0.843** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.084 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.030 | **0.946** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.024 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LM | 0.000 | 0.000 | 0.000 | 0.010 | 0.000 | 0.000 | 0.039 | **0.907** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.044 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LP | 0.000 | 0.000 | 0.000 | 0.014 | 0.000 | 0.000 | 0.009 | **0.115** | **0.676** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.186** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.033 | 0.000 | 0.000 | 0.067 | 0.000 | 0.000 | **0.233** | **0.233** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.433** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SC | **0.096** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.074 | **0.649** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.181** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.053 | **0.947** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.708** | 0.000 | 0.001 | **0.152** | 0.000 | 0.067 | 0.000 | 0.065 | 0.000 |
| Cm | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.019 | 0.000 | **0.687** | **0.096** | **0.193** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.058 | **0.882** | 0.054 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.042 | 0.087 | **0.764** | **0.099** | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.014 | **0.971** | 0.013 | 0.000 | 0.000 | 0.000 |
| Ob | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.024 | 0.024 | **0.951** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.029 | 0.000 | 0.000 | 0.006 | **0.964** | 0.000 |

**Table 10** Change probabilities of design anti-patterns and design patterns in all studied systems

| | Source | AS | Bl | CS | CC | LC | LZC | LM | LP | MCh | RP | SC | SG | SA | Bu | Cm | Cp | FM | De | Ob | PT | Si | Sink |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AS | 0.022 | **0.961** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.010 | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 |
| Bl | 0.001 | **0.309** | **0.378** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.092 | 0.000 | 0.000 | 0.000 | 0.000 | **0.218** |
| CS | 0.006 | 0.000 | 0.008 | **0.971** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.008 | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 |
| CC | 0.000 | 0.000 | 0.000 | 0.029 | **0.936** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.019 | 0.000 | 0.000 | 0.000 | 0.000 | 0.014 |
| LC | 0.000 | 0.000 | 0.000 | 0.000 | **0.5** | **0.5** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| LZC | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.011 | **0.979** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 |
| LM | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.028 | **0.937** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.015 | 0.000 | 0.000 | 0.000 | 0.000 | 0.012 |
| LP | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.000 | 0.002 | 0.022 | **0.929** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.013 | 0.000 | 0.000 | 0.000 | 0.000 | 0.028 |
| MCh | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| RP | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | **0.115** | **0.770** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.052 | 0.000 | 0.000 | 0.000 | 0.000 | 0.056 |
| SC | 0.031 | 0.000 | 0.000 | 0.031 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.049 | **0.811** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.102** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SG | 0.000 | 0.000 | 0.000 | 0.005 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.010 | 0.015 | **0.947** | 0.000 | 0.000 | 0.000 | 0.000 | 0.015 | 0.000 | 0.000 | 0.000 | 0.000 | 0.005 |
| SA | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Bu | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.150** | 0.000 | 0.003 | **0.509** | 0.000 | **0.323** | 0.000 | 0.009 | 0.000 |
| Cm | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.004 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | **0.146** | **0.239** | **0.566** | 0.017 | 0.021 | 0.000 | 0.000 | 0.000 |
| Cp | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.005 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.072 | **0.838** | 0.078 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| FM | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.001 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.023 | **0.111** | **0.814** | 0.043 | 0.000 | 0.000 | 0.000 | 0.000 |
| De | 0.000 | 0.001 | 0.001 | 0.002 | 0.000 | 0.002 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **0.144** | **0.734** | **0.108** | 0.000 | 0.000 | 0.000 |
| Ob | 0.015 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.095 | **0.126** | **0.761** | 0.000 | 0.000 | 0.000 |
| PT | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | **1** | 0.000 | 0.000 |
| Si | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.013 | 0.000 | 0.000 | 0.013 | **0.966** | 0.006 |

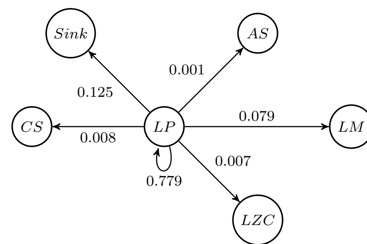


**Fig. 7** LongParameterList (LP) mutation in ApacheSolr.

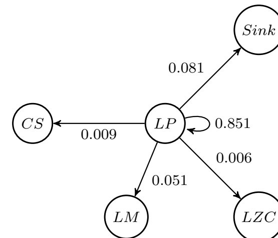


**Fig. 8** LongParameterList (LP) mutation in ApacheIgnite.

Tables 3 to 9 show the mutations between design patterns and design anti-patterns that occurred in their evolutions. Table 10 aggregates all the mutations in all the systems. We added two additional states (source and sink) to describe the appearances of design (anti-)patterns (sources) and the disappearance of some design (anti-)patterns (sinks).

The mutation probabilities shown in previous tables are percentages that may hide outliers. Therefore, we also calculate the standard deviation values among these probabilities. We found that the probabilities across snapshots have low standard-deviation values, as shown in Table 11, with the highest value of 0.196 for Nuxeo.

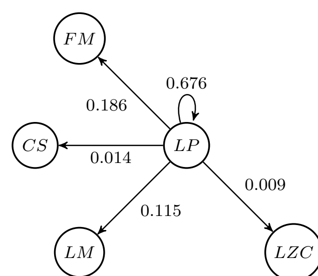


**Fig. 9** LongParameterList (LP) mutation in Mule.

**Table 11** Standard deviation values and confidence levels

| Systems | Standard Deviation | Confidence Level | Margin of Error |
|---|---|---|---|
| Apache Ignite | 0.1966 | 90%, 1.645 $\sigma x$ | 0.04347±0.0147 (±33.87%) |
| Apache Solr | 0.1921 | 90%, 1.645 $\sigma x$ | 0.04343±0.0144 (±33.12%) |
| Eclipse | 0.1833 | 90%, 1.645 $\sigma x$ | 0.04317±0.0137 (±31.79%) |
| Matsim | 0.1722 | 90%, 1.645 $\sigma x$ | 0.04304 ±0.0129 (±29.96%) |
| Mule | 0.1766 | 90%, 1.645 $\sigma x$ | 0.04345 ±0.0132(±30.43%) |
| Nuxeo | 0.1968 | 90%, 1.645 $\sigma x$ | 0.0451 ±0.0147 (±32.67%) |
| oVirt | 0.1961 | 90%, 1.645 $\sigma x$ | 0.04352 ±0.0147 (±33.73%) |

**Table 12** Mean values of the mutations of design anti-patterns and design pattern occurrences mutated in all the snapshots of each system

| Systems | Mean Value of DAPs mutations | Mean Value of DPs mutations |
|---|---|---|
| Apache Ignite | 0.0799 | 0.0037 |
| Apache Solr | 0.1026 | 0.0232 |
| Eclipse | 0.1355 | 0.1128 |
| Matsim | 0.2013 | 0.1917 |
| Mule | 0.1989 | 0.1341 |
| Nuxeo | 0.0848 | 0.03 |
| oVirt | 0.0674 | 0.0252 |

Table 11 shows a systematic analysis of the confidence levels of our results. We computed the standard-deviation values and confidence intervals of our results for a confidence level of 90% as follows:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{1}$$

where $x_i$ is each value from the population (mutations probabilities), $\mu$ is the mean of the population, and $N$ is the size of the population, *i.e.,* the total number of mutations in all the snapshots of a system.

With the standard deviation known, we compute the confidence interval for a population mean as:

$$\bar{X} \pm Z \times \frac{\sigma}{\sqrt{N}} \tag{2}$$

where $\bar{X}$ is plus or minus a margin of error, $Z$ is the Z-value for the chosen confidence level, $\sigma$ is a standard deviation and $N$ is the size of the population.

We observe that, for a confidence level of 90%, the confidence intervals, which indicate how much we can expect the results to reflect the observations from the overall population, were around 30% in all the analysed systems. We consider these values of confidence levels and intervals acceptable to deduce trends and infer conclusions. Indeed, while we could not find similar discussions and numbers in other software-engineering papers, we observed similar values used to deduce trends in other domains, *e.g.,* public health [46].

For example, SpaghettiCode has the most representative mutation probability from Source in Mule (see Table 9) and Blob to Sink in Apache Solr (see

**Table 13** Most representative mutations between design patterns and design anti-patterns according to their mutation probabilities

| System | Mutation Type | From | To | Probability |
|---|---|---|---|---|
| Apache Ignite | DAP → DAP | Blob (Bl) | AntiSingleton (AS) | 0.375 |
| | DAP → DP | - | - | - |
| | DP → DAP | - | - | - |
| | DP → DP | Builder (Bu) | Observer (ob) | 0.004 |
| Apache Solr | DAP → DAP | Blob (Bl) | AntiSingleton (AS) | 0.321 |
| | DAP → DP | - | - | - |
| | DP → DAP | - | - | - |
| | DP → DP | FactoryMethod (FM) | Composite (Cp) | 0.012 |
| Eclipe IDE | DAP → DAP | LargeClass (LC) | ComplexClass (Cc) | 0.500 |
| | DAP → DP | - | - | - |
| | DP → DAP | FactoryMethod (FM) | LongMethod (LM) | 0.003 |
| | DP → DP | FactoryMethod (FM) | Composite (Cp) | 0.169 |
| Matsim | DAP → DAP | Blob (Bl) | AntiSingleton (AS) | 0.372 |
| | DAP → DP | Blob (Bl) | FactoryMethod (FM) | 0.346 |
| | DP → DAP | Command (Cm) | SwissArmyKnife (SA) | 0.030 |
| | DP → DP | Command (Cm) | FactoryMethod (FM) | 0.387 |
| Mule | DAP → DAP | Blob (bl) | AntiSingleton(AS) | 0.313 |
| | DAP → DP | RefusedParentBequest (RP) | FactoryMethod (FM) | 0.433 |
| | DP → DAP | Command (Cm) | SwissArmyKnife (SA) | 0.019 |
| | DP → DP | Command (Cm) | FactoryMethod | 0.193 |
| Nuxeo | DAP → DAP | Blob (bl) | AntiSingleton(AS) | 0.283 |
| | DAP → DP | Blob (Bl) | FactoryMethod (FM) | 0.297 |
| | DP → DAP | Singleton (Si) | LazyClass (ZC) | 0.004 |
| | DP → DP | Singleton (Si) | FactoryMethod (FM) | 0.133 |
| ovirt | DAP → DAP | Blob (bl) | AntiSingleton(AS) | 0.299 |
| | DAP → DP | - | - | - |
| | DP → DAP | Singleton (Si) | AntiSingleton (AS) | 0.001 |
| | DP → DP | Singleton (Si) | Prototype (PT) | 0.097 |

Table 7). Table 13 shows the most representative design patterns and design anti-patterns regarding mutation probabilities to/from other patterns.

**Analysing pattern evolution:** We observe in Table 13 that not all design patterns and design anti-pattern undergo changes. Some patterns remain stable during evolution. For example, LazyClass and MessageChain are stable design anti-patterns, while Prototype is a persistent design pattern in Apache Solr. In Matsim, design anti-patterns SwissArmyKnife, LazyClass, and MessageChain and design patterns Observer and ProtoType are stable.

However, in general, design anti-patterns tend to evolve in all studied systems. We observe that more than half of the design anti-patterns mutated into other design patterns or design anti-patterns across the different snapshots of the studied system.

For example, in Matsim (see Table 6), 86% (probability value 0.86) LongMethod remains stable and mutate with only a probability of 14% into other patterns. In oVirt (see Table 5), Blob remain persistent in the system with 39.4% probability, while 29.9% mutated to AntiSingleton and into other patterns with a probability of 30.8%. As last example, in Eclipse, 45.5% of RefusedParentBequest remain between snapshot while 27.3% mutated to MessageChain and 27.3% mutated into other patterns.

We saw fewer mutations among design patterns. As an example, in Apache Ignite, 97.6% of Command remained stable, with only 0.4% mutating into other patterns.

For a better understanding of the design pattern and design anti-pattern mutations, Figures 3 to 9 show the most representative mutations in the

Markov models as graphs, for each of the systems. Because showing all possible probabilities would make the graphs unreadable, we choose a threshold of 0.100 to reduce the number of edges in each graph. The gray cells in Tables 3 to 9 highlight the probabilities shown in the figures. (The graphs for all the mutations in all the systems are available on-line[8].)

These graphs contain information on mutating design (anti-)patterns that can help developers to avoid the mutations that could have negative impacts on software quality.

**Summary:** Our results show that design patterns and design anti-patterns mutate during the evolution of software systems. Despite an average of less than 20% of the identified design anti-patterns occurrences having mutated among all the snapshots of systems, more than half of the types of design anti-patterns were involved in these mutations. For example, Table 12 shows that, in Matsim, 20.13% of the design anti-pattern identified in the first snapshot mutated during the studied period. During that period, ten types of design anti-patterns out of the 13 considered in our study were involved in mutations. In most of the systems, almost all the design patterns remained stable during evolution. Blob and Command are the design anti-patterns and design pattern with the higher mutation probabilities.

### 5.2 **RQ2:** *What types of changes lead to mutations between design patterns and design anti-patterns?*

**Motivation:** Knowing the causes of design (anti-)patterns mutations would be useful during maintenance. They could help developers to focus on the most frequent change types triggering patterns mutations.
**Analysing change types:** We use srcML[9] to create an XML file for each snapshot of a system and match their tags to find changed tags, as explained in Section 3.5. We categorise change types based on our categories in Table 1. We compare the percentages of each change type for each system. We apply the same methodology for all the subject systems. Figures 10 to 16 show the types of changes per design (anti-)patterns per systems.

Table 14 presents the numbers of each change types in all the systems. For example, in Apache Ignite, we observe that Access and Renaming are the least and most representative change types for both design patterns and design anti-patterns. They lead to many changes in occurrences of both design anti-patterns and design patterns.

[8] http://www.ptidej.net/downloads/replications/emse19c/
[9] https://www.srcml.org/

**Table 14** Number of different types of changes in design patterns and design anti-patterns

| Systems → | Eclipse | | Nuxeo | | oVirt | | Matsim | | Apache Solr | | Apache Ignite | | Mule | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Change Types ↓** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| | **DAP** | **DP** | **DAP** | **DP** | **DAP** | **DP** | **DAP** | **DP** | **DAP** | **DP** | **DAP** | **DP** | **DAP** | **DP** |
| Access | 33 | 6 | 85 | 0 | 174 | 9 | 271 | 117 | 34 | 15 | 11 | 55 | 20 | 12 |
| Class | 268 | 91 | 236 | 6 | 3804 | 90 | 1697 | 690 | 721 | 155 | 781 | 218 | 443 | 198 |
| Code block | 4197 | 1075 | 1070 | 13 | 13923 | 233 | 8805 | 3203 | 3873 | 459 | 3903 | 203 | 1430 | 413 |
| Comment | 32939 | 11388 | 9269 | 109 | 15013 | 411 | 22519 | 9287 | 9298 | 2616 | 14554 | 1567 | 6354 | 3780 |
| Control Flow | 10487 | 1870 | 966 | 10 | 6440 | 118 | 5398 | 2094 | 3166 | 636 | 3433 | 133 | 916 | 384 |
| Declaration | 9721 | 2789 | 3133 | 24 | 27605 | 487 | 24244 | 9609 | 8803 | 1392 | 9527 | 349 | 3904 | 1103 |
| Exception | 996 | 341 | 946 | 1 | 619 | 29 | 1602 | 314 | 1696 | 457 | 2076 | 64 | 505 | 173 |
| Import | 2566 | 835 | 2734 | 23 | 18819 | 211 | 13013 | 4679 | 4024 | 491 | 4584 | 394 | 3234 | 793 |
| Invocation | 1707 | 394 | 556 | 4 | 7312 | 91 | 8520 | 3026 | 2598 | 287 | 2069 | 75 | 945 | 240 |
| Method | 4060 | 942 | 1487 | 29 | 13792 | 292 | 4702 | 1922 | 3215 | 511 | 3364 | 266 | 1940 | 747 |
| Operator | 13540 | 2803 | 3533 | 8 | 35513 | 403 | 57112 | 24326 | 7963 | 702 | 7207 | 525 | 5241 | 1975 |
| Parameter | 5629 | 1541 | 2179 | 3 | 24488 | 292 | 22080 | 5149 | 8375 | 1069 | 9024 | 332 | 3252 | 756 |
| Renaming | 59254 | 14707 | 16259 | 23 | 262491 | 3536 | 294661 | 145720 | 44422 | 4811 | 63961 | 4396 | 28110 | 9738 |
| **#Changed classes** | 10957 | 3155 | 5402 | 81 | 34780 | 55 | 32596 | 13768 | 7956 | 1192 | 9290 | 857 | 5684 | 2175 |
| **Total classes** | 20331 | 7574 | 39051 | 1263 | 142537 | 2482 | 62272 | 79480 | 32332 | 5490 | 27080 | 5796 | 47146 | 17553 |

**AP, DP**= Number of changes in design anti-patterns and design patterns respectively

**Table 15** Number of different types of changes in design patterns and design anti-patterns mutation.

| Systems→ | Eclipse | | Nuxeo | | oVirt | | Matsim | | Apache Solr | | Apache Ignite | | Mule | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Change Types ↓** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| | **APDP** | **DPAP** | **APDP** | **DPAP** | **APDP** | **DPAP** | **APDP** | **DPAP** | **APDP** | **DPAP** | **APDP** | **DPAP** | **APDP** | **DPAP** |
| Access | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 12 | 3 | 0 | 0 | 0 | 0 | 1 |
| Class | 3 | 1 | 2 | 0 | 10 | 0 | 4 | 29 | 1 | 7 | 2 | 4 | 6 | 3 |
| Code block | 1 | 2 | 1 | 4 | 21 | 22 | 10 | 132 | 4 | 1 | 1 | 3 | 27 | 17 |
| Comment | 102 | 192 | 7 | 4 | 69 | 31 | 38 | 739 | 129 | 242 | 58 | 23 | 160 | 85 |
| Control Flow | 28 | 38 | 0 | 3 | 7 | 6 | 12 | 96 | 44 | 31 | 16 | 4 | 21 | 3 |
| Declaration | 56 | 78 | 3 | 2 | 82 | 38 | 56 | 412 | 141 | 90 | 33 | 16 | 49 | 32 |
| Exception | 4 | 6 | 0 | 0 | 4 | 1 | 6 | 28 | 19 | 13 | 14 | 4 | 3 | 2 |
| Import | 20 | 14 | 3 | 2 | 32 | 18 | 28 | 257 | 60 | 34 | 17 | 16 | 28 | 18 |
| Invocation | 6 | 7 | 0 | 0 | 15 | 20 | 7 | 183 | 1 | 6 | 5 | 6 | 10 | 3 |
| Method | 16 | 18 | 2 | 2 | 41 | 24 | 7 | 96 | 64 | 43 | 10 | 12 | 23 | 13 |
| Operator | 38 | 37 | 2 | 0 | 43 | 66 | 95 | 523 | 22 | 20 | 6 | 13 | 87 | 32 |
| Parameter | 22 | 41 | 0 | 0 | 37 | 20 | 16 | 423 | 25 | 47 | 22 | 29 | 13 | 22 |
| Renaming | 675 | 469 | 3 | 2 | 297 | 1036 | 462 | 5096 | 165 | 406 | 100 | 63 | 334 | 280 |
| #Changed classes | 71 | 77 | 7 | 6 | 61 | 54 | 58 | 681 | 58 | 66 | 34 | 29 | 53 | 39 |

**APDP, DPAP**= Number of changes in design anti-patterns to design patterns and design patterns to design anti-patterns mutations respectively

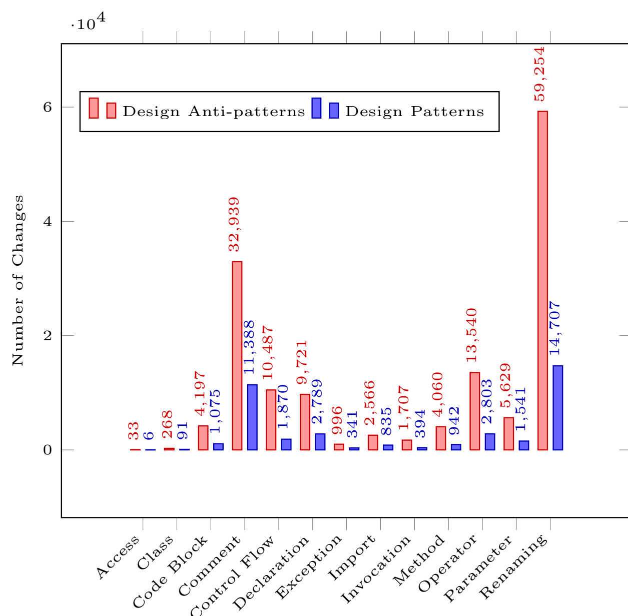


**Fig. 10** Number of different types of changes in Eclipse classes with design anti-patterns and design patterns.

**Analysing change types of mutations:** During evolution, design patterns and design anti-patterns can mutate into other design patterns and design anti-patterns. We investigate which types of changes lead to such mutations. Tables 15 shows the number of each change types during the mutation for all the studied systems.

Results show that, in Apache Ignite, Renaming, Comment, and Declaration lead the most mutations from design anti-patterns (DAPs) to design patterns (DPs). It is almost the same for DPs-to-DAPs mutations but Parameter has more importance than Declaration. In Apache Solr and Eclipse for both DAPs-to-DPs and DPs-to-DAPs mutations, Renaming, Declaration, and Comment are the most representative change types. In Matsim, Renaming, Operator, and Declaration have the most impact on DAPs-to-DPs mutations while Renaming, Comment, and Operator lead to more DPs-to-DAPs mutations. In Mule, for both DAPs-to-DPs and DPs-to-DAPs mutations, Renaming, Comment, and Operator are the most representative change types. In Nuxeo, there are few mutations, in which Comment, Renaming, and Declaration yield DAPs-to-DPs mutations while Comment, Code Block, and Control Flow yield more DPs-to-DAPs mutations. Finally, in oVirt, Renaming, Declaration, and Comment are change types that lead to DAPs-to-DPs mutations while Renaming, Operator, and Declaration bring DPs-to-DAPs mutations.

Renaming is the most frequent change type. There are different types of renaming, described in [47]. In some types, an entity, *e.g.,* a package, a class,

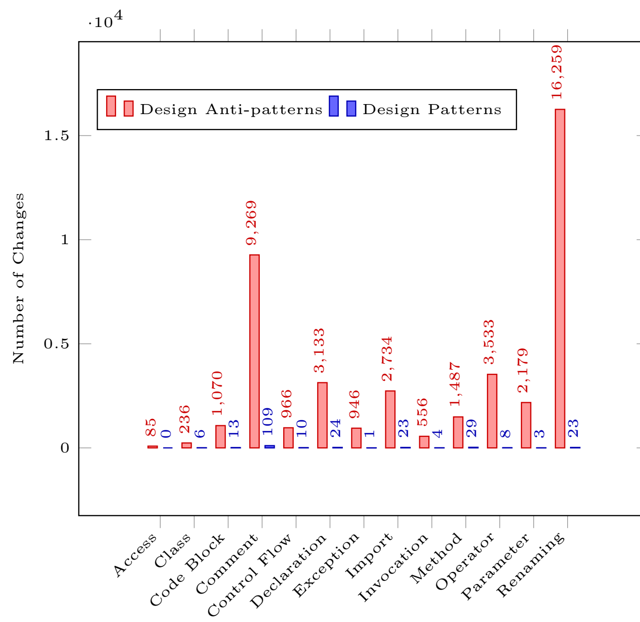


**Fig. 11** Number of different types of changes in Nuxeo classes with design anti-patterns and design patterns.

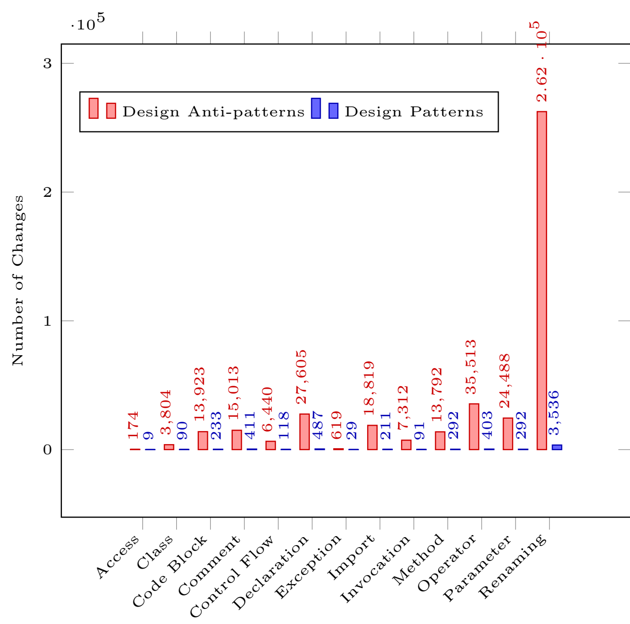


**Fig. 12** Number of different types of changes in oVirt classes with design anti-patterns and design patterns.

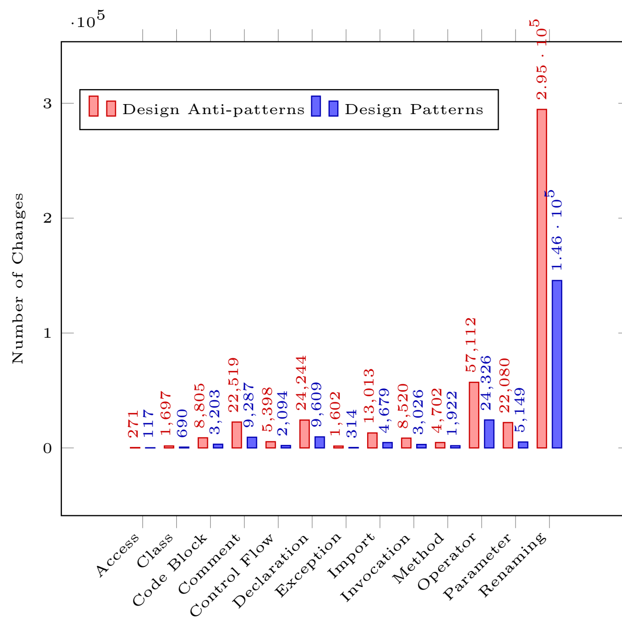


**Fig. 13** Number of different types of changes in Matsim classes with design anti-patterns and design patterns.

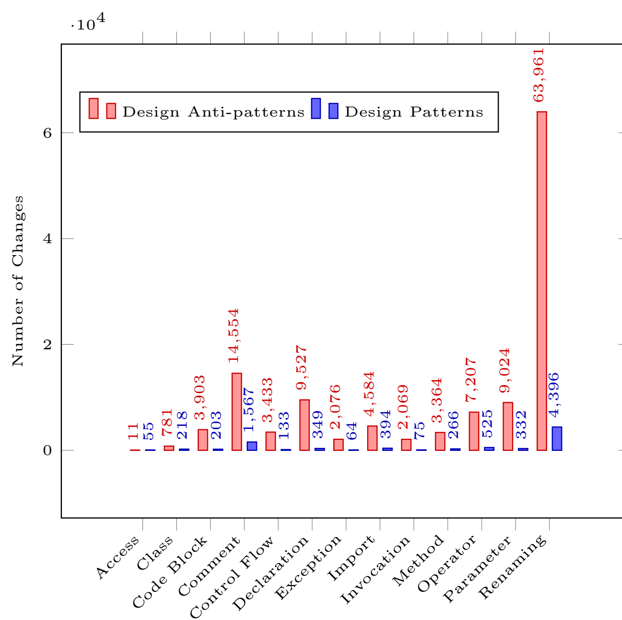


**Fig. 14** Number of different types of changes in Apache Ignite classes with design anti-patterns and design patterns.

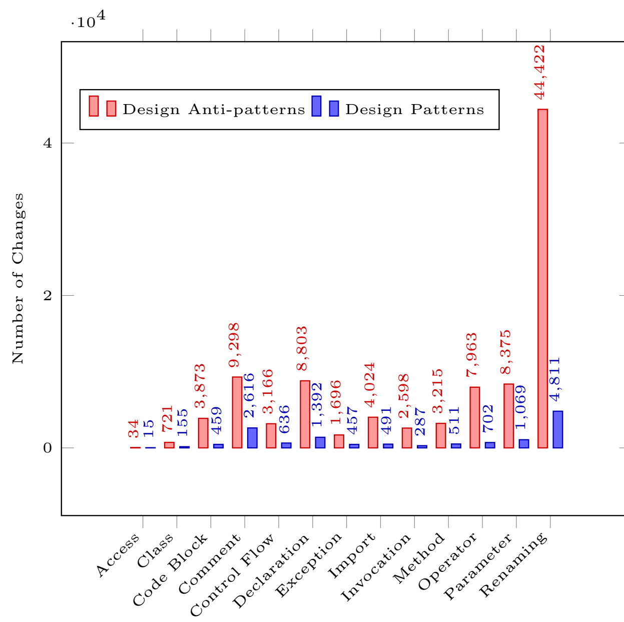


**Fig. 15** Number of different types of changes in Apache Solr classes with design anti-patterns and design patterns.

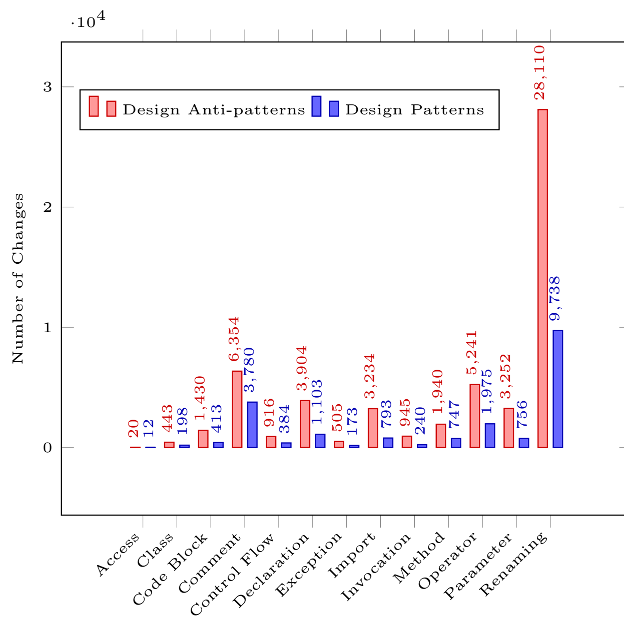


**Fig. 16** Number of different types of changes in Mule classes with design anti-patterns and design patterns.

etc., is renamed. In other types, one or more terms are changed (simple and complex renaming). Sometimes, when one or more terms change, the meaning of the identifier also changes (semantic renaming). The grammar of an identifier may also change during evolution (grammar renaming). When developers use some tools to apply a renaming operation, the tool may not rename all variables consistently in all related files. Besides, certain source-code changes must be made together to preserve consistency. Changes in comments and declarations led to more mutations not as root causes of these mutations but because developers changed comments and declarations while evolving their systems for other reasons, *i.e.,* fixing faults. Future work include a manual, qualitative analysis of the mutations to identify their root causes.

**Summary:** *Some change types affect the mutations between design patterns and–or design anti-patterns more than others. We observe that the change types leading to mutations in all the studied systems are Renaming, Comment, Declaration, and Operator.*

### 5.3 **RQ3:** *What is the fault-proneness of mutated design patterns and anti-patterns and what transitions lead to more fault-prone mutations?*

**Motivation:** The results from RQ1 and RQ2 show that design patterns and design anti-patterns mutate during software evolution. However, they do not say anything about the impact of these mutations in software quality. Therefore, we investigate these mutations and their fault-proneness. Using this information, developers could understand the reasons of faults and take actions to reduce the risk of introducing faults.

**Analysing design patterns and design anti-patterns fault-proneness:** For each system, we mine its commit log and extract fault and commit IDs related to the faults and the dates when the faults were introduced. We look for faults introduced from one snapshot to the next. We use the dates to distinguish between faults appearing in one snapshot and those appearing between two extracted snapshots.

Table 16 shows the numbers of faulty and clean of classes involved in patterns. One class can be involved in several faults in the studied snapshots so we show in Table 17 the numbers of unique faulty and clean classes involved in patterns. Finally, in Table 18, we compare the percentages of faulty and clean classes involved in design patterns and design anti-patterns. Moreover, we show in Table 18 the relative percentages of faults per class participating, or not, in design (anti-)patterns.

With these tables, we summarise our whole dataset with: the numbers of (unique) faulty and non-faulty (clean) classes participating or not in design (anti-)patterns and their relative percentages. We thus can compare the prevalence of faults in different classes and confirm that classes participating in design anti-patterns have more faults than classes involved in design pat-

terns. Thus, we have evidence supporting that classes that participate in design anti-patterns are more fault-prone than classes involved in design patterns.

For example, in Eclipse, 13.7% of design pattern classes are fault-prone while 37.3% of design anti-pattern classes are fault-prone. Table 18 is showing similar trends in all the analysed systems.

**Table 16** Design anti-pattern and design-pattern mutations between faulty and clean classes

| **Systems** | **# of Faulty classes** | | **# of Clean classes having DAPs, DPs** |
|---|---|---|---|
| | Design Anti-patterns | Design Patterns | |
| Apache Ignite | 10,984 | 1,051 | 81,093 |
| Apache Solr | 11,156 | 219 | 109,225 |
| Eclipse | 15,240 | 5,182 | 19,928 |
| Matsim | 4,053 | 1,888 | 896,510 |
| Mule | 17,794 | 5,924 | 197,574 |
| Nuxeo | 18,724 | 396 | 146,180 |
| oVirt | 12,605 | 110 | 217,565 |

**Table 17** Faulty and clean classes

| **Systems** | **# of Faulty Classes** | | | | **# of Faulty Classes** | | **# of Clean Classes** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | DAPs | | DPs | | **without Patterns** | | DAPs | | DPs | |
| Apache Ignite | 10,984 | (685) | 1,051 | (84) | 3,984 | (3,984) | 71,784 | (3,474) | 9,309 | (395) |
| Apache Solr | 11,156 | (638) | 219 | (22) | 6,351 | (6,351) | 101,044 | (3,447) | 8,181 | (392) |
| Eclipse | 15,240 | (591) | 5,182 | (217) | 12,285 | (12,285) | 13,610 | (562) | 6,318 | (213) |
| Matsim | 4,053 | (469) | 1,888 | (115) | 291 | (291) | 326,460 | (14,425) | 570,050 | (9,913) |
| Mule | 17,794 | (2,955) | 5,924 | (196) | 4,370 | (4,370) | 126,980 | (7,341) | 70,594 | (1,593) |
| Nuxeo | 18,724 | (1,487) | 396 | (68) | 7,414 | (7,414) | 143,276 | (3,659) | 2,904 | (170) |
| oVirt | 12,605 | (377) | 110 | (8) | 523 | (523) | 214,027 | (7,653) | 3,538 | (214) |

**Table 18** Faulty and clean classes in percentages

| **Systems** | **Design anti-patterns** | | **Design patterns** | |
|---|---|---|---|---|
| | Faulty Classes (%) | Clean Classes (%) | Faulty Classes (%) | Clean Classes (%) |
| Apache Ignite | 14.7% | 74.9% | 1.8% | 8.5% |
| Apache Solr | 14.1% | 76.6% | 0.48% | 8.7% |
| Eclipse | 37.3% | 35.5% | 13.7% | 13.4% |
| Matsim | 1.9% | 57.9% | 0.46% | 39.7% |
| Mule | 24.4% | 60.7% | 1.6% | 13.2% |
| Nuxeo | 27.6% | 67.9% | 1.3% | 3.2% |
| oVirt | 4.6% | 92.7% | 0.09% | 2.6% |

**Analyzing mutations fault-proneness:** A mutation between design patterns and design anti-patterns can lead to faults. We use clean and faulty classes and their participation (or not) into design patterns and design anti-patterns to identify the mutations experienced by these faulty classes.

Table 19 presents the most representative mutations that led to faults in each studied system. We observe that mutations from design anti-patterns to other design anti-patterns are more faulty. LongParameterList to LongMethod or LongMethod to LazyClass are such mutations in Apache Ignite.

In Eclipse, Matsim, and Mule, there are mutations from design patterns to design patterns that also led to more faults. FactoryMethod to Decorator in Eclipse, Builder to FactoryMethod in Matsim and Mule are such mutations.

There are also mutations from design anti-patterns to design patterns that led to faults as well, like AntiSingleton to FactoryMethod in Matsim or FactoryMethod to LongMethod in Eclipse.

**Table 19** Most representative mutations between design patterns and design anti-patterns according to their mutation probabilities and fault-proneness

| System | Mutation Type | From | To | Probability |
|---|---|---|---|---|
| Apache Ignite | DAP → DAP | LongParameterList | LongMethod | 0.571 |
| | DAP → DAP | LongMethod | LazyClass | 0.285 |
| Apache Solr | DAP → DAP | RefusedParentBequest | MessageChain | 0.427 |
| | DAP → DAP | LongMethod | LazyClass | 0.156 |
| | DAP → DAP | ComplexClass | ClassDataShouldBePrivate | 0.156 |
| Eclipe IDE | DP → DP | FactoryMethod | Decorator | 0.492 |
| | DAP → DAP | LongMethod | LazyClass | 0.385 |
| | DP → DAP | FactoryMethod | LongMethod | 0.056 |
| Matsim | DP → DP | Builder | FactoryMethod | 0.677 |
| | DAP → DAP | SpagettiCode | RefusedParentBequest | 0.152 |
| | DAP → DP | AntiSingleton | FactoryMethod | 0.114 |
| Mule | DP → DP | Builder | FactoryMethod | 0.479 |
| | DAP → DP | ComplexClass | FactoryMethod | 0.264 |
| | DAP → DAP | ComplexClass | ClassDataShouldBePrivate | 0.223 |
| Nuxeo | DAP → DAP | LazyClass | LargeClass | 0.285 |
| | DP → DP | Singleton | FactoryMethod | 0.495 |
| oVirt | DAP → DAP | Blob | AntiSingleton | 0.722 |
| | DP → DP | Singleton | Prototype | 0.166 |

**Summary:** *We observed that in some systems, as expected and shown in previous work, design anti-patterns are more fault-prone than design patterns. We also showed that some mutations are more fault-prone than others, in particular mutations from design anti-patterns to design patterns or to other design anti-patterns.*

### 5.4 **RQ4:** *Do specific types of changes lead to increased fault-proneness during mutations?*

**Motivation:** Different types of changes have different impacts on the software systems due to their differences in functionality and the ripple effects of changes. Some types of changes likely introduce more faults than others. Thus, understanding which types of changes increase the fault-proneness of the mutations could help developers to foresee and prevent faults by preventing/planning such changes during software evolution.

**Analysing change types leading to faults:** We use the same data as in RQ2 and RQ3. For each system, we identify the number of faulty classes that have changed through mutations between design anti-patterns and–or design patterns. Table 20 shows the number of change types that led to faults. We report

that, in all studied systems, Renaming, Comment, and Operator are the change types that lead to more faults.

**Table 20** Numbers of change types in the studied systems leading to faults

| Systems → | Apache Ignite | Apache Solr | Eclipse | Matsim | Mule | Nuxeo | oVirt |
|---|---|---|---|---|---|---|---|
| **Change Types** | # changes | # changes | # changes | # changes | # changes | # changes | # changes |
| Access | 18 | 18 | 37 | 22 | 11 | 62 | 25 |
| Class | 689 | 431 | 306 | 306 | 422 | 208 | 763 |
| Code block | 2,972 | 2,102 | 4,919 | 1,284 | 1,306 | 854 | 3,833 |
| Comment | 12,169 | 5,498 | 38,150 | 2,813 | 6,270 | 6,161 | 4,653 |
| Control flow | 2,903 | 1,935 | 11,678 | 660 | 1067 | 819 | 2,406 |
| Declaration | 5,912 | 5,386 | 11651 | 3,129 | 3,191 | 2,628 | 7,484 |
| Exception | 1,696 | 1221 | 1255 | 140 | 526 | 786 | 210 |
| Import | 2,831 | 2,400 | 2,958 | 1,425 | 2,443 | 2,064 | 4,268 |
| Invocation | 1,550 | 1,196 | 1,986 | 1,061 | 840 | 476 | 1,882 |
| Method | 2,509 | 1,851 | 4,093 | 637 | 1,697 | 1,229 | 3,619 |
| Operator | 5,120 | 4,364 | 16,094 | 6,215 | 4,228 | 2,675 | 8,134 |
| Parameter | 6,418 | 3,504 | 5,108 | 2,655 | 2,607 | 1,815 | 5,337 |
| Renaming | 47,811 | 24,640 | 67,040 | 32,445 | 22,968 | 12,736 | 65,245 |
| **Total changed classes** | 5,505 | 4,163 | 11,934 | 3,073 | 4,324 | 3,514 | 7,150 |

**Analyzing fault-proneness of classes with design patterns and design anti-patterns:** Table 21 presents the numbers of faulty changed classes and Figures 17 and 18 show the percentages of faulty changed classes participating in design patterns and design anti-patterns for all the systems. Figure 19 compares the numbers of faulty and clean classes changed in all the snapshots of the studied systems. Each first bar presents the number of faulty and clean changed classes participating in design anti-patterns and the second one is faulty and clean classes having design patterns. We observe that change types have impacts on the fault-proneness of changed classes. Changed classes participating in design patterns are less faulty than those participating only in design anti-patterns.

Figures 17 and 18 show that some of the faulty classes are those which had changed in the past. For example, in Eclipse, the percentages of faulty classes participating in design patterns is 81% while for those participating in design anti-patterns it is 86%. The differences between these two categories are more visible in Apache Solr, where 51% of changed classes are participating in design anti-patterns, and only 11% of them have design patterns. In Rhino, changes impact fault-proneness significantly, because, on average, more than 85% of changed classes are faulty. Thus, the trend is that changed classes with design anti-patterns tend to be more fault-prone than changed classes with design patterns.

**Summary:** *Some change types applied to design patterns and design anti-patterns make software systems more fault-prone compared to others. We observed that, in all the studied systems, Renaming, Comment, and Operator are the change types from design patterns to design anti-patterns that most lead to faults.*

**Table 21** Numbers of faulty and clean changed classes

| Systems | Patterns | # Faulty classes | # Clean classes |
|---|---|---|---|
| Apache Ignite | Design Anti-patterns | 5,112 | 4,178 |
| | Design Patterns | 393 | 464 |
| Apache Solr | Design Anti-patterns | 4,035 | 3,921 |
| | Design patterns | 128 | 1,064 |
| Eclipse | Design Anti-patterns | 9,406 | 1,551 |
| | Design patterns | 2,554 | 601 |
| Matsim | Design Anti-patterns | 2,549 | 30,042 |
| | Design patterns | 524 | 13,244 |
| Mule | Design Anti-patterns | 3,374 | 2,311 |
| | Design patterns | 950 | 1,225 |
| Nuxeo | Design Anti-patterns | 3,469 | 1,935 |
| | Design patterns | 45 | 36 |
| oVirt | Design Anti-patterns | 7,075 | 27,705 |
| | Design patterns | 75 | 482 |

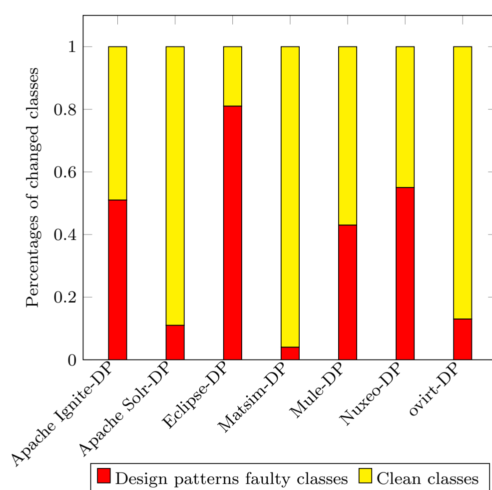


**Fig. 17** Faulty changed classes percentages with design pattern in the studied systems

## 6 Discussion

By comparing the results between all studied systems, we observed remarkable differences in the proportions and types of mutations of design anti-patterns and design patterns. For example, in oVirt in Table 5, 29.9% of Blob mutated to AntiSingleton (the highest mutation percentages). However, we did not observe in the same system any mutation from Blob to a design pattern. On the contrary, in Nuxeo, although the mutation of Blob to AntiSingleton remains frequent (28.3%), the highest mutation proportion observed is Blob to Factory Method (29.7%).

Another remarkable observation is that, in all studied systems, the occurrences of the design anti-pattern SwissArmyKnife and of the design pattern

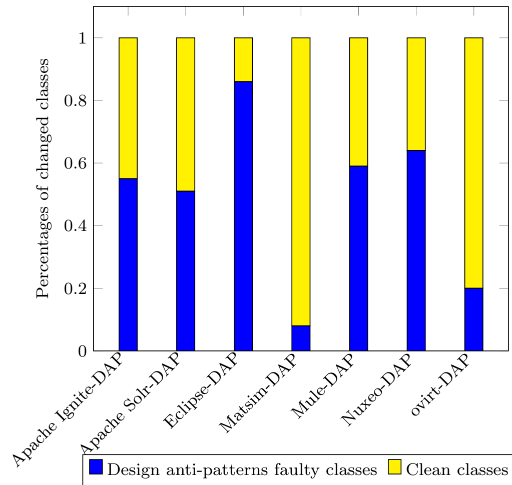


**Fig. 18** Faulty changed classes with design anti-patterns percentages in the studied systems

Prototype never mutated. We cannot draw any conclusion about a potential impact of the mutation of these patterns on the quality of the systems.

Different software systems may have different design patterns and–or design anti-patterns and may evolve differently. From our analysis, we observe different mutation behavior for all analyzed systems because these systems have different designs, contexts, and development teams. We observed that some design patterns and–or design anti-patterns remained unchanged in all releases and they did not mutate during evolution.

For example, class *org.mule.test.infrastructure.process.MuleUtils* in all the snapshots of Mule, is a LongMethod design anti-pattern. This design anti-pattern is introduced when developers continue adding new functionalities to a method while nothing is never taken out. Usually, developers prefer to add code to an existing method instead of creating a new one [15], which means that another line is added and then another, giving birth to a tangle of spaghetti code. This longer method or function become harder to understand and maintain.

We found that some of the design anti-patterns are mutated frequently to design patterns when developers are correcting faults during the evolution of the systems. Blob is the most mutated design anti-pattern in Apache Ignite. It mutated to AntiSingleton with 37.5% probability. Blob presents a single class with a large number of attributes, operations, etc., surrounded by a number of data classes. A Blob is too complex for reuse and testing, while such classes are inefficient, and expensive to load into memory.

There are also some design patterns that mutated to design anti-patterns. Command is an example of a design pattern that often mutated into SwissArmyKnife (38.7% of the time) in Matsim. SwissArmyKnife is an excessively complex class interface. Developers attempted to provide for all possible uses of the class. They added a large number of interfaces (APIs) to meet all pos-

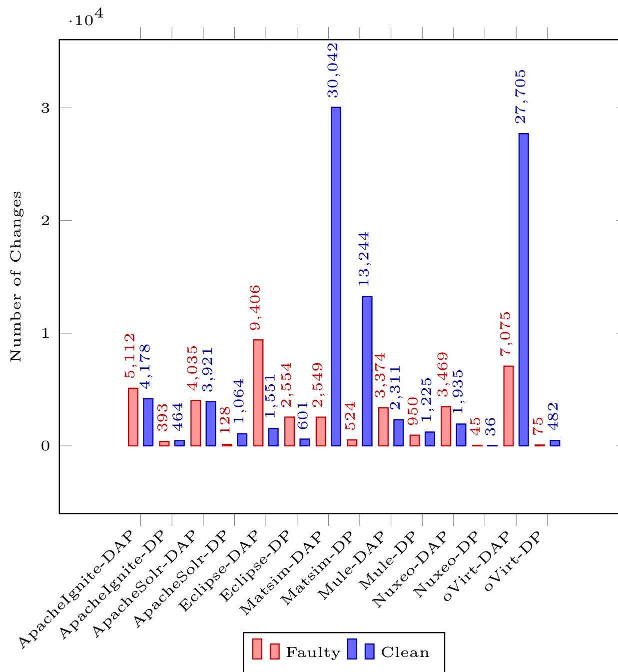


**Fig. 19** Faulty changed classes with design anti-patterns and design patterns in the studied systems

sible needs. The code to create separate Command classes grow to encompass more functionalities and become a SwissArmyKnife.

We observed that the change types that led to more mutations are Renaming, Comment, Operator, and Declaration, in most of the studied systems. These types of changes helped developers to correct their systems and remove some design anti-patterns. We also noticed that the most frequent mutation was LongParameterList to LongMethod.

We found in some analysed systems, that design anti-patterns are more fault-prone than classes involved in design patterns, while in some other systems, it is quite the opposite. Although these observations mean that employing design patterns may not always benefit the software quality, and introducing anti-patterns will not systemically compromise the software quality, some previous research related similar observations.

Bieman *et al.* [28] observed that large classes participating to design patterns were more change-prone. Vokač *et al.* [12] found a significant differences in the fault-proneness of different design patterns. Gatrell *et al.* [29] observed that pattern-based classes are more change-prone and less stable than non-pattern classes [12]. Long [48] showed the benefits of some anti-patterns in the context of code reuse.

The context of using design anti-patterns or design patterns [48], their static and historical relationships [11,49], and their internal characteristics,

such as their numbers of lines of code [28], could be compounding factors that lead to our previous observations, which confirm previous findings that software quality decreases with design anti-patterns and increases with design patterns.

A fault is an error that causes a software system to produce an incorrect or unexpected result or to behave in unintended ways. Although several previous works, *e.g.,* [7, 14], showed that an important proportion of faults may be related to design patterns, anti-patterns, and mutations among them, many other faults remain unrelated to patterns or their mutations. Table 17 shows the numbers of (unique) faults per classes participating or not in some design (anti-)patterns. It shows that, in all studied systems except Matsim, the numbers of classes with faults but not participating in any design (anti-)pattern is higher than the number of classes with faults and participating in some design (anti-)pattern.

The results in this paper generally confirm results from previous studies and provide new research directions to further understand the impact of programming practices, like patterns and evolution, on software quality. First, while we confirmed that, generally, classes participating in design anti-patterns have more faults than classes participating in design patterns or no patterns; we also reported contradictory cases, like the design anti-pattern SwissArmyKnife and the design pattern Prototype. Thus, further studies are needed to understand the root causes of such cases, through manual, qualitative analyses of the changes leading to the faults.

We showed that some mutations are more frequent than others between some design anti-patterns and design patterns. We reported changes that led, concretely, to these mutations. However, our study is only on co-occurrences of these changes and mutations: more studies are required to understand why some patterns mutate into others in some systems but not others. We believe that these mutations and their differences may be due to the systems themselves and their developers but further studies are required to identify root causes and assert causation.

Finally, we showed that some mutations are more fault prone than others, which could guide developers but also researchers. For developers, our observations can help avoiding harmful changes, *i.e.,* fault-prone mutations, and focus some refactoring activities on beneficial changes. For researchers, our observations provide a first step in understanding the introduction of patterns and the changes that may lead to them as well as their impact on fault-proneness.

## 7 Threats to Validity

We now discuss potential threats to the validity of the results of our study, following existing guidelines [50, 51].

**Construct validity:** These threats concern the relation between theory and observation. We know that the used design pattern and design anti-pattern detection techniques (DECOR and DeMIMA) in this study include some sub-

jective understanding related to the definition of design patterns and design anti-patterns. Their authors reported recall rates of 100% for both techniques while the precision in the worst case was 31%. We accept that the precision of these techniques is a concern. Some false positive classes may pass the validation because they "looks like" playing a role in some patterns.

We also accept that, in finding change types which led to faults, we could have matched classes that are not representing the actual same class. For example, class `C` is not match with `a.b.C.java` but could be matched with the `b.c.java`. Moreover, we know that during evolution, class names change as well. As for precision, the manual validation could be affected by subjectiveness or human error. We should consider each type of renaming as we may misinterpret that there is a mutation between design patterns and design anti-patterns, while in fact the class name changed and the patterns remained stable.

**Internal validity:** This threat concerns factors affecting our results. This threat is about the causality drawn from the study. It concerns our selection of studied systems and methodology. The accuracy of DECOR and DeMIMA impacts our results, because the number of design patterns and design anti-patterns computed with DECOR and DeMIMA is used to calculate the probabilities of mutations. Other detection techniques should be used to validate our findings.

Our results show correlations between design anti-patterns and design patterns, their mutations, and faults. However, they do not show causation. Hence, it is possible that some of the changes, which led to mutations, *e.g.,* changes to comments, although correlated to mutations, are not the root causes of these mutations. Identifying these root causes would require studying each change leading to mutations individually, manually, which is future work.

**Conclusion validity:** These threats concern the relationship between the treatment and the results. We paid attention in choosing the systems.

We used the SZZ algorithm [19] to identify commits introducing faults. Although this algorithm may yield false positive results, it has been successfully employed in previous works, such as [52, 53]. In this paper, to increase the algorithm's accuracy, we removed all fault-inducing commit candidates that only changed blank or comment lines. Moreover, the static analysis tool, srcML, can identify about 100 types of code elements from source code.

To make our results more actionable for software practitioners, we manually grouped similar element tags into 12 major change types as shown in Table 1, which can help developers carefully change and review fault-prone code.

**Reliability Validity:** These threats concern the possibility of replicating the study. We provide all the necessary data on-line[10] to help other researchers replicate our work.

**External validity:** These threats concern the ability to generalize our results. We studied seven software systems with different sizes, domains, and complexity. We selected only Java systems because of the tools. We also chose some of these systems because they have been used in previous studies. Their

[10] `http://www.ptidej.net/downloads/replications/emse19c/`

numbers of lines of code range from hundred of thousands to several millions. These systems are widely used and have active developers community. They have several years of evolution histories. They are available on-line. However, all of them are written in Java and are open source. In the future, we plan to investigate more diverse set of systems. Moreover, we also want to study larger projects, with other programming languages, such as C++.

We analysed commits instead of releases to cover as much as possible the whole histories of these systems. We choose thirteen design anti-patterns and eight design patterns among the many available patterns.

## 8 Conclusion and Future Work

We investigated the evolution and impacts of design patterns and design anti-patterns in terms of change- and fault-proneness during software evolution. We built Markov models to analyse the mutations of design patterns and design anti-patterns in seven open-source Java systems: Apache Ignite, Apache Solr, Eclipse, Matsim, Mule, Nuxeo, and Ovirt. We identified the change types that led to mutations and we calculated the probabilities of all possible mutations. Finally, we reported which patterns are mostly mutated into which other patterns (including appearance and disappearance) as well as change types.

Results showed that design patterns and design anti-patterns mutate into one another during software evolution and that these mutations impact the fault-proneness of classes participating in these patterns. Generally, when a mutation led to the introduction of a design anti-pattern, quality in terms of fault-proneness decreased; when a design anti-pattern was removed or mutated into a design pattern, quality increased.

Using this information, developers can focus on the design patterns that are most likely to mutate into design anti-patterns and–or to have more faults. Thus, this information can help evolution and quality assurance by focusing refactoring efforts on classes with design (anti-)patterns that could mutate into patterns with higher fault-proneness.

In future work, we intend to apply our study on more systems written in different programming languages and also consider more design (anti-)patterns. We will also attempt to identify the root causes of mutations by studying manually and qualitatively the changes leading to mutations.

## References

1. E. Gamma, *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.
2. I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, “Code quality analysis in open source software development,” *Information Systems Journal*, vol. 12, no. 1, pp. 43–60, 2002.

3. F. Khomh and Y.-G. Guéhéneuc, "Perception and reality: What are design patterns good for?" in *Proceedings of 11th ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering (QAOOSE), Springer-Verlag*, 2007, p. 7.
4. M. Fowler and K. Beck, *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 1999.
5. E. Van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on.* IEEE, 2002, pp. 97–106.
6. M. Mäntylä, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on.* IEEE, 2003, pp. 381–384.
7. F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change-and fault-proneness," *Empirical Software Engineering*, vol. 17, no. 3, pp. 243–275, 2012.
8. M. Abbes, F. Khomh, Y.-G. Gueheneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on.* IEEE, 2011, pp. 181–190.
9. A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, "A methodology to assess the impact of design patterns on software quality," *Information and Software Technology*, vol. 54, no. 4, pp. 331–346, 2012.
10. F. Khomh and Y.-G. Guéhéneuc, "Do design patterns impact software quality positively?" in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on.* IEEE, 2008, pp. 274–278.
11. F. Jaafar, Y.-G. Guéhéneuc, and S. Hamel, "Analysing anti-patterns static relationships with design patterns," *Proc. PPAP*, vol. 2, p. 26, 2013.
12. M. Vokáč, "Defect frequency and design patterns: An empirical study of industrial code," *Software Engineering, IEEE Transactions on*, vol. 30, no. 12, pp. 904–917, 2004.
13. L. Aversano, G. Canfora, L. Cerulo, C. Del Grosso, and M. Di Penta, "An empirical study on the evolution of design patterns," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering.* ACM, 2007, pp. 385–394.
14. F. Jaafar, F. Khomh, Y.-G. Guéhéneuc, and M. Zulkernine, "Anti-pattern mutations and fault-proneness," in *Quality Software (QSIC), 2014 14th International Conference on.* IEEE, 2014, pp. 246–255.
15. W. H. Brown, R. C. Malveau, and T. J. Mowbray, *AntiPatterns: refactoring software, architectures, and projects in crisis.* Wiley, 1998.
16. N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *Software Engineering, IEEE Transactions on*, vol. 36, no. 1, pp. 20–36, 2010.
17. Y.-G. Guéhéneuc and G. Antoniol, "Demima: A multilayered approach for design pattern identification," *Software Engineering, IEEE Transactions on*, vol. 34, no. 5, pp. 667–684, 2008.
18. S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability.* Springer Science & Business Media, 2012.
19. J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *ACM sigsoft software engineering notes*, vol. 30, no. 4. ACM, 2005, pp. 1–5.
20. B. F. Webster, *Pitfalls of object oriented development.* M\ & T Books, 1995.
21. A. J. Riel, *Object-oriented design heuristics.* Addison-Wesley Reading, 1996, vol. 335.
22. R. Marinescu and M. Lanza, "Object-oriented metrics in practice," 2006.
23. D. Settas, A. Cerone, and S. Fenz, "Enhancing ontology-based antipattern detection using bayesian networks," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9041–9053, 2012.
24. S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, "A meta-model for language-independent refactoring," in *Proceedings International Symposium on Principles of Software Evolution.* IEEE, 2000, pp. 154–164.
25. S. Ducasse, T. Gîrba, and R. Marinescu, "Using history information to improve design flaws detection," in *CSMR 2004: 8TH EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING.* Citeseer, 2004.

26. C. Kramer and L. Prechelt, "Design recovery by automated search for structural design patterns in object-oriented software," in *Reverse Engineering, 1996., Proceedings of the Third Working Conference on.* IEEE, 1996, pp. 208–215.
27. C. Iacob, "A design pattern mining method for interaction design," in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems.* ACM, 2011, pp. 217–222.
28. J. M. Bieman, G. Straw, H. Wang, P. W. Munger, and R. T. Alexander, "Design patterns and change proneness: An examination of five evolving systems," in *Software metrics symposium, 2003. Proceedings. Ninth international.* IEEE, 2003, pp. 40–49.
29. M. Gatrell, S. Counsell, and T. Hall, "Design patterns and change proneness: a replication using proprietary c# software," in *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on.* IEEE, 2009, pp. 160–164.
30. S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement.* IEEE Computer Society, 2009, pp. 390–400.
31. A. F. Yamashita and L. Moonen, "Do developers care about code smells? an exploratory survey." in *WCRE*, vol. 13, 2013, pp. 242–251.
32. S. E. S. Taba, F. Khomh, Y. Zou, A. E. Hassan, and M. Nagappan, "Predicting bugs using antipatterns," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on.* IEEE, 2013, pp. 270–279.
33. Y.-G. Gueheneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design patterns," in *Reverse Engineering, 2004. Proceedings. 11th Working Conference on.* IEEE, 2004, pp. 172–181.
34. M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on.* IEEE, 2003, pp. 23–32.
35. L. An and F. Khomh, "An empirical study of crash-inducing commits in mozilla firefox," in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering.* ACM, 2015, p. 5.
36. "srcML," http://www.srcml.org, 2016, online; Accessed March 31st, 2016.
37. D. Romano, P. Raila, M. Pinzger, and F. Khomh, "Analyzing the impact of antipatterns on change-proneness using fine-grained source code changes," in *Reverse Engineering (WCRE), 2012 19th Working Conference on.* IEEE, 2012, pp. 437–446.
38. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *Software Engineering, IEEE Transactions on*, vol. 32, no. 11, pp. 896–909, 2006.
39. J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Reading: Addison-Wesley*, vol. 49, no. 120, p. 11, 1995.
40. F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "Playing roles in design patterns: An empirical descriptive and analytic study," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on.* IEEE, 2009, pp. 83–92.
41. M. Lanza and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems.* Springer Science & Business Media, 2007.
42. D. Rapu, S. Ducasse, T. Gîrba, and R. Marinescu, "Using history information to improve design flaws detection," in *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on.* IEEE, 2004, pp. 223–232.
43. S. Vaucher, F. Khomh, N. Moha, and Y.-G. Guéhéneuc, "Tracking design smells: Lessons from a study of god classes," in *16th Working Conference on Reverse Engineering (WCRE 2009), IEEE Computer Society Press (WCRE'09)*, 2009.
44. A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering.* IEEE Computer Society, 2009, pp. 78–88.
45. G. Canfora, L. Cerulo, M. Di Penta, and F. Pacilio, "An exploratory study of factors influencing change entropy," in *2010 IEEE 18th International Conference on Program Comprehension.* IEEE, 2010, pp. 134–143.

46. P. Strazzullo, L. DElia, N.-B. Kandala, and F. P. Cappuccio, "Salt intake, stroke, and cardiovascular disease: meta-analysis of prospective studies," *Bmj*, vol. 339, p. b4567, 2009.
47. V. Arnaoudova, L. M. Eshkevari, M. Di Penta, R. Oliveto, G. Antoniol, and Y.-G. Gueheneuc, "Repent: Analyzing the nature of identifier renamings," *IEEE Transactions on Software Engineering*, vol. 40, no. 5, pp. 502–532, 2014.
48. J. Long, "Software reuse antipatterns," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 4, pp. 68–76, 2001.
49. F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and F. Khomh, "Mining the relationship between anti-patterns dependencies and fault-proneness," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 351–360.
50. R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.
51. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
52. Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.
53. T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 172–181.