A Safari for Deviating GoF Pattern Definitions and Examples on the Web

Apostolos V. Zarras(✉) and Panos Vassiliadis

Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
zarras@cs.uoi.gr

Abstract. The Gang of Four (GoF) patterns have been around for many years now. People use them to solve object-oriented design problems. The main source to consult for the GoF patterns is the seminal book published by Gamma, Helm, Johnson, and Vlissides in 1994. However, today there is also a large amount of information about the GoF patterns on the Web. There, the developers can find pattern definitions and code examples.

In this paper, we assess the compliance of pattern definitions and examples found on the Web to the original GoF pattern definitions. We study a corpus of definitions and examples, gathered from 4 well-known sites. According to our findings, most of the provided pattern definitions comply with the original GoF pattern definitions. However, there are some intent deviations that result in incorrect definitions. There are also a few deviations that concern missing and incomplete participants. When it comes to the patterns examples, the situation is quite different. Deviations in the examples are much more frequent and include missing participants, incomplete participants, and erroneous participants. The paper concludes with a discussion of the practical implications of our findings for the developers.

Keywords: GoF design patterns · deviating definitions · deviating examples

1 Introduction

In the mid-nineties, Gamma, Helm, Johnson and Vlissides introduced the Gang of Four (GoF) patterns catalog [8]. The GoF catalog documents reusable object-oriented solutions to common development problems. The authors employ a unified form for the specification of the patterns. A pattern specification includes the name of the pattern, the intent of the pattern, a motivating scenario, a discussion of the pattern applicability, a diagrammatic description of the pattern structure, the responsibilities of the participants (i.e., classes or interfaces) involved in the pattern structure, a description of how the participants collaborate to carry out their responsibilities, the consequences of the pattern, implementation guidelines, sample code, known uses of the pattern, and, a discussion concerning other related patterns. The patterns are divided in three different categories:

J. P. A. Almeida et al. (Eds.): ER 2023, LNCS 14320, pp. 181–197, 2023.
https://doi.org/10.1007/978-3-031-47262-6_10

creational patterns that deal with the creation of objects, structural patterns that concern the composition of classes or objects and behavioral patterns that focus on the interaction between objects and the distribution of responsibilities.

Nowadays, the GoF catalog is a significant part of object-oriented design theory and practice. Many people like students, junior developers and more experienced developers search on the Web for information about the GoF patterns, for a variety of reasons. Popular Web sites provide definitions of the patterns and examples that illustrate instances of the patterns in specific contexts. The patterns definitions range from brief statements of the patterns intent to more detailed ones that specify the patterns structure and other details.

The story behind this paper starts a couple of years ago, when an undergraduate student came to complain about his grade on a software engineering quiz. Specifically, the student claimed that his answer to a question about a design pattern was correct. While discussing the issue, the student said that he had studied the pattern very well and that he had also used in his study additional sources that he found on a Web site. When we looked again at the information given about the pattern on that site we found out that both the definition and the examples of the pattern deviated from the original GoF pattern definition. In fact, we discovered that several examples were entirely wrong!

So, sometimes the information we find on the Web about the GoF patterns deviates from the original definitions of the patterns that are given in the GoF catalog.

This observation is the main motivation of this paper. The overall **research goal** of the paper is **to assess the compliance of pattern definitions and examples that we find on the Web to the original GoF pattern definitions**. To this end, we study a corpus of pattern definitions and examples, gathered from four different well-known Web sites. At first, we identify the **kinds of deviations** that occur in the **pattern definitions** and **examples**. To highlight the issues that arise from the different kinds of deviations, we discuss in detail characteristic examples. Then, we study the **amount of deviations**, the **percentage** of deviating definitions and examples, and the **density** of deviations in the patterns definitions and examples. Finally, we discuss the implications of our findings for the developers who seek information about GoF patterns on the Web.

The rest of this paper is structured as follows. In Sect. 2 we discuss related work. In Sect. 3 we provide details regarding the setup of this study. In Sect. 4, we report the different kinds of deviations that occur in pattern definitions and examples. In Sect. 5, we assess the compliance of the pattern definitions and examples to the original GoF patterns. In Sect. 6, we conclude with the practical implications of our findings.

2 Related Work

Design patterns have been an active area of research and practice for decades. The state of the art is vast with dedicated communities and venues for researchers

and practitioners[1,2]. An interesting systematic mapping of the state of the art is provided by Mayvan et al. [10]. According to this study, research efforts in the area of design patterns can be divided in 5 major sub-areas that concern pattern development, specification, usage, mining, and quality evaluation.

Our study falls in the area of pattern quality evaluation. This area includes efforts that investigate the impact of pattern usage on software quality and efforts that assess the soundness of pattern instances.

The impact of design pattern usage in software quality has been the subject of several interesting empirical studies. For instance, Prechelt et al. [12] found that the use of design patterns provides flexibility, which facilitates maintenance. In another study, Prechelt et al. [11] observed that the use of design patterns, along with specialized comments related to these patterns is helpful towards performing maintenance tasks. According to Vokac [14], the use of patterns by itself does not guarantee few defects. Bieman et al. [4] observed that pattern classes are prone to changes. Aversano et al. [3] observed that pattern classes are more prone to changes than classes that depend on the pattern classes. Walter and Alkhaeir [15] and Alfadel et al. [1] observed that classes participating to design patterns are less prone to code smells than classes not participating to design patterns. A detailed systematic literature review of works that concern the relation between design patterns and code smells has been performed by Almadi et al. [2].

Regarding the soundness of design patterns, Izurieta and Bieman [9] introduced the notions of design pattern rot and grime. Design rot is the breakdown of the structural integrity of a design pattern instance, as a result of changes in subsequent software releases. Design pattern grime is a decay due to unrelated features added in classes that participate in a design pattern instance. These features do not jeopardize the intent of the pattern. In a study of 3 software systems, Izurieta and Bieman did not find evidence of design rot. However, they found evidence of design pattern grime. In further studies, Dale and Izurieta [5] observed that certain kinds of design grim results in higher technical debt, while Reimanis and Izurieta [13] investigated possible correlations between different kinds of grime. In a study of five software systems, Feitosa et al. [7] found a linear accumulation of design grime that depends on the design patterns and the developers, while in a subsequent study [6] the same authors observed correlations between the accumulation of grime and decreased performance, security and correctness.

Still, regarding the soundness of design patterns, Zarras [17] observed frequent mistakes in the usage of the Command pattern, during the project of a software engineering course. The observed mistakes concern the configuration of command objects and invalidate the benefits of the Command pattern. The author further introduced a pattern for the proper configuration of command objects. In a similar effort, Zarras [16] reported mistakes in the usage of Strategy pattern and introduced respective solutions in the form of patterns.

[1] hillside.net.
[2] www.europlop.net.

Going beyond the state of the art, this paper evaluates the compliance of patterns definitions and examples found on the Web to the original GoF pattern definitions in a study that involves a large number of definitions and examples gathered from different sites.

3 Setup of the Study

Our study considers four popular Web sites that provide information for several software development topics like refactoring, UML, design principles and patterns. In particular, we focus on pattern definitions and examples from Source Making[3], Refactoring Guru[4], Tutorials Point[5] and Java T Point[6].

Table 1. Corpus of patterns definitions and examples.

		Source Making		Refactoring Guru		Tutorials Point		Java T Point		ALL	
Corpus Patterns' Definitions and Examples		Definition	Example(s)	Definition	Example(s)	Definition	Example(s)	Definition	Example(s)	Definitions	Examples
Creational	Abstract Factory	✓	7	✓	10	✓	1	✓	1	4	19
	Builder	✓	5	✓	10	✓	1	✓	1	4	17
	Factory Method	✓	6	✓	10	✓	1	✓	1	4	18
	Prototype	✓	7	✓	10	✓	1	✓	1	4	19
	Singleton	✓	4	✓	10	✓	1	✓	1	4	16
Sructural	Adapter	✓	6	✓	10	✓	1	✓	1	4	18
	Bridge	✓	5	✓	10	✓	1	✓	1	4	17
	Composite	✓	9	✓	10	✓	1	✓	1	4	21
	Decorator	✓	8	✓	10	✓	1	✓	1	4	20
	Façade	✓	5	✓	10	✓	1	✓	1	4	17
	Flyweight	✓	7	✓	10	✓	1	✓	1	4	19
	Proxy	✓	6	✓	10	✓	1	✓	1	4	18
Behavioral	Chain of Responsibility	✓	6	✓	10	✓	1	✓	1	4	18
	Command	✓	8	✓	10	✓	1	✓	1	4	20
	Interpreter	✓	5	×	0	✓	1	✓	1	3	7
	Iterator	✓	6	✓	10	✓	1	✓	1	4	18
	Mediator	✓	6	✓	10	✓	1	✓	1	4	18
	Memento	✓	5	✓	10	✓	1	✓	1	4	17
	Observer	✓	8	✓	10	✓	1	✓	1	4	20
	State	✓	9	✓	10	✓	1	✓	1	4	21
	Strategy	✓	5	✓	10	✓	1	✓	1	4	17
	Template Method	✓	5	✓	10	✓	1	✓	1	4	17
	Visitor	✓	6	✓	10	✓	1	×	0	3	17
	Total	23	144	22	220	23	23	22	22	90	409

During the data-gathering process we visited each Web site. For each GoF pattern, we looked for the pattern definition and examples that illustrate the usage of the pattern. We downloaded local copies of the pattern definition and examples. At the end of the gathering process, we reviewed the retrieved definitions and examples to make sure that we did not omit any relevant definitions and examples.

[3] sourcemaking.com.
[4] refactoring.guru.
[5] www.tutorialspoint.com/design_pattern.
[6] www.javatpoint.com/design-patterns-in-java.

The corpus of our study consists of 90 pattern definitions and 409 examples that concern a variety of programming languages. The corpus and the raw data of the compliance evaluation are available online[7]

Table 1 provides further details concerning the corpus. More specifically, Source Making provides definitions for all of the GoF patterns and a total number of 144 examples. There is at least one example for every GoF pattern. Most of the examples are in Java and C++. However, there are also several examples in Delphi, PhP and Python. Refactoring Guru covers all, but the Interpreter pattern. Therefore the corpus includes 22 definitions and 220 examples. The examples are in Pseudo code, Java, C#, C++, PhP, Python, Ruby, Swift, Typescript, and Go. Tutorials Point and Java T Point focus only on Java. Tutorials Point covers all of the GoF patterns, while Java T Point covers all but the Visitor pattern. Thus, Tutorials Point and Java T Point add 23 and 22 patterns definitions and examples to the corpus, respectively.

To assess the compliance of a set of pattern definitions (respectively examples) to the original GoF pattern definitions we rely on three basic statistics:

- The *number of deviations* that occur in the examined set.
- The *percentage* of *deviating pattern definitions* (respectively, *examples*) in the examined set.
- The *density of deviations* in the examined set, defined as the number of deviations, over the cardinality of the examined set.

In all our deliberations, the diagrams that we use are made by us, for copyright purposes, with respect to the diagrams that accompany the GoF pattern definitions, the diagrams that accompany the pattern definitions of the sites and the source code of the pattern examples given in the sites.

The deviation analysis that concerns *pattern definitions* relies on the combination of text and diagrams of the site contrasted to the GoF definition text and diagrams. The comparison protocol involves the following sequence of checks:

- The first check concerns whether the intent of the pattern definition given in the site is inline with the intent of the GoF pattern definition.
- The second check concerns whether the participants specified in the GoF pattern definition are present in the pattern definition of the site.
- The third check concerns whether the participants specified in the pattern definition of the site provide the methods of the corresponding participants of the GoF pattern definition.

The deviation analysis that concerns the *pattern examples* is based on the text and the source code of the pattern examples given in the sites, contrasted to the GoF definition text and diagrams. The comparison protocol involves the following steps:

[7] Due to their volume, data are available in a non-monitored, anonymous google drive (https://drive.google.com/file/d/1vAn58ul7whaXM01TMFcj1er5UcCSzrYN/view?usp=sharing), to become eponymously public at github upon acceptance of the paper.

- The first check concerns whether the participants specified in the GoF pattern definition are present in the pattern example.
- The second check concerns whether the participants in the pattern example provide the methods of the corresponding participants of the GoF pattern definition.
- The third check concerns whether the implementation of the participants in the pattern example is inline with the behavior of the corresponding participants of the GoF pattern definition.

4 Kinds of Deviations

In the corpus, we identified four different kinds of deviations. Specifically, we found intent deviations, missing participants, incomplete participants and erroneous participants. In the rest of this section, we discuss each kind of deviations in more detail, along with respective examples.

Table 2. Intent deviations found in the corpus.

Web site	Pattern	Deviating intent	Original Intent
Tutorials Point	**Abstract Factory**	**"Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern."**	**"Provide an interface for creating families of related or dependent objects without specifying their concrete classes."**
Tutorials Point	**Builder**	**"Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. A Builder class builds the final object step by step. This builder is independent of other objects."**	**"Separate the construction of a complex object from its representation so that the same construction process can create different representations."**
Tutorials Point	**Visitor**	**"In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object."**	**"Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates."**
Java T Point	**Observer**	**"An Observer Pattern says that "just define a one-to-one dependency so that when one object changes state, all its dependents are notified and updated automatically."**	**"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically."**

4.1 Intent Deviations

In the corpus, we identified certain pattern definitions that do not reflect the purpose of the corresponding patterns, as specified in the GoF catalog. Hereafter, we call these issues, **intent deviations**. Obviously an intent deviation is very important, as it always results in an incorrect definition. Table 2, illustrates the deviating pattern definitions that we found in the corpus. Specifically, the table gives the Web site that provides each pattern definition, the deviating intent of the pattern, and the original intent of the pattern, as specified in the GoF catalog.

At a glance, in Tutorials Point the intent of Abstract Factory is defined quite differently from the original definition of the pattern. According to the original GoF definition, an abstract factory is an interface that defines methods for creating families of related objects, without having to specify their concrete classes, while according to the Tutorials Point definition the abstract factory is an interface that defines methods for creating other factories.

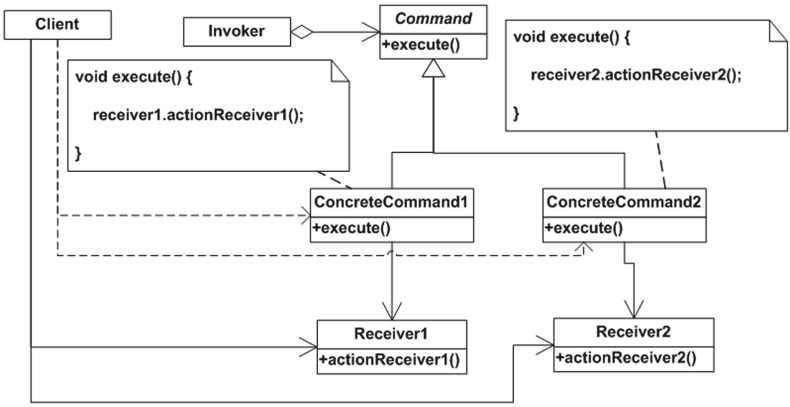

Fig. 1. Command structure, as defined in the GoF catalog.

In the Tutorials Point definition of Builder, the main issue is that there is absolutely no mention of the separation between the construction process of a complex object and the different representations of the object, which is the key benefit of the pattern. The intent of Visitor in the Tutorials Point definition is also very different from the original definition. According to the Tutorials Point definition, the purpose of a visitor class is to change the algorithm of another class, while in the latter the focus is on extensibility, and specifically the addition of new operations that operate on an hierarchy of objects, without having to change this hierarchy. Finally, in the Java T Point definition of Observer the intent of the pattern is incorrect as it refers to one-to-one, instead of one-to-many, dependencies between subscribers and observers.

4.2 Missing Participants

In the corpus, we encountered pattern definitions and examples that do not include all the participants, specified in the structure of the original pattern definitions. Hereon, we use the term **missing participants** to refer to these participants. The criticality of missing participants depends on the pattern and on who these participants are. In some cases, missing participants may result in incorrect pattern definitions and examples. In other cases, missing participants may result in incomplete pattern definitions and examples that partially illustrate the original pattern concepts.

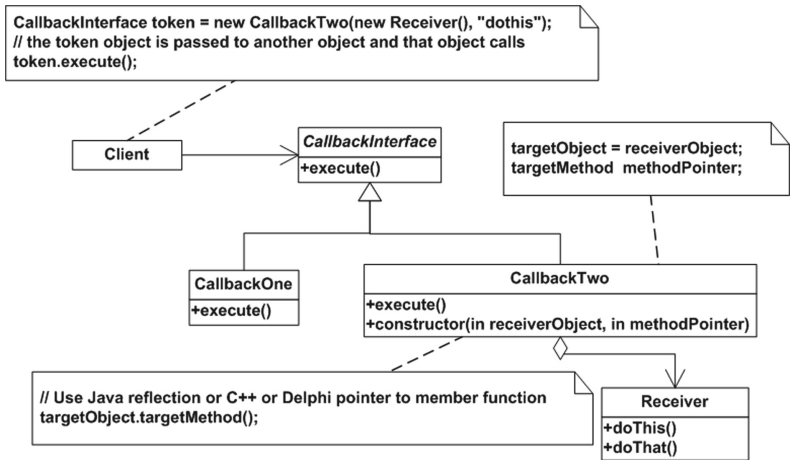

Fig. 2. Command structure, as defined in Source Making.

Figure 1, gives the structure of the Command pattern, as specified in the GoF patterns catalog. The intent of Command is to *"encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations"*. `Command` defines a common interface for executing different commands. `ConcreteCommand1` and `ConcreteCommand2` are different classes that implement the `Command` interface. `Client` creates `Command` objects that belong to the different implementation classes. `Invoker` is parameterized with `Command` objects. To execute a command, `Invoker` invokes the `execute()` method on a particular `Command` object.

Figure 2, gives the structure of the Command pattern, as defined in Source Making. In particular, `Client`, `CallbackInterface`, `CallbackOne` and `CallbackTwo`, correspond to `Client`, `Command`, `ConcreteCommand1` and `ConcreteCommand2`, in the original pattern structure, respectively. Apparently, in the Source Making definition, the `Invoker` participant is missing. The `Client` participant creates `Command` objects and invokes the `execute()` method to

execute the corresponding commands. The lack of `Invoker` is important here because the pattern definition does not reflect the concept of parameterization of objects with different commands.

4.3 Incomplete Participants

The corpus includes patterns definitions and examples involving **incomplete participants** that do not provide a complete and exact set of methods, as they should according to the original pattern definitions. Specifically, some methods may be missing, or some methods may be merged with others in larger methods that have more responsibilities than they should.

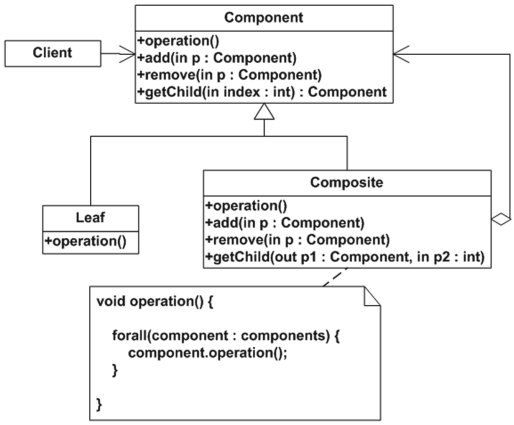

Fig. 3. Composite structure, as defined in the GoF catalog.

The impact of incomplete participants depends on the methods that are actually missing. In some cases, the lack of certain methods is very important (e.g. the lack of certain creation methods in creational patterns), resulting in incorrect pattern definitions and examples, while in other cases the missing methods result in incomplete definitions and examples that partially illustrate the concepts of the original pattern.

Figure 3, shows the original structure of the Composite pattern. The purpose of the pattern is to *"compose objects into tree structures to represent part-whole hierarchies"*. `Component` is a class that defines a uniform interface for both primitive and composite objects. The interface includes domain-specific methods (like `operation()`) and methods for managing the structure of composite objects. Specifically, `add()` serves for adding a `Component` object to a `Composite` object, while `remove()` allows removing a `Component` object from the `Composite` object. The `getChild()` method allows retrieving a `Component` object that is part of the `Composite` object, based on a given index. The interesting point in the pattern is that `Component` not only defines the uniform interface, but also provides default

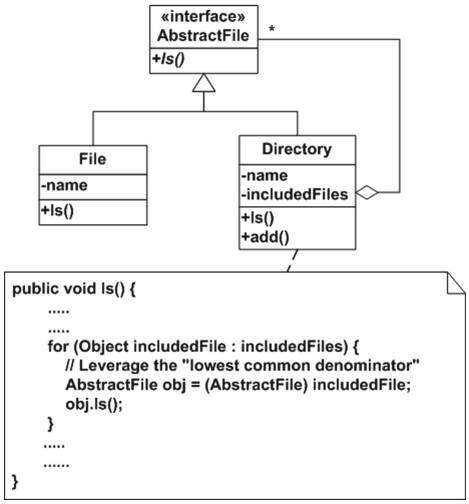

Fig. 4. Example of Composite from Source Making.

implementations for the defined methods. `Leaf` represents primitive objects that do not consist of other objects. `Leaf` provides its own implementations for the domain-specific methods and inherits the default implementations of the structure management methods, defined in `Component`. `Composite` represents composite objects. It provides implementations for both the domain-specific methods and the structure management methods, defined in `Component`. `Client` manipulates objects that conform with the aforementioned composite structure, via the uniform `Component` interface.

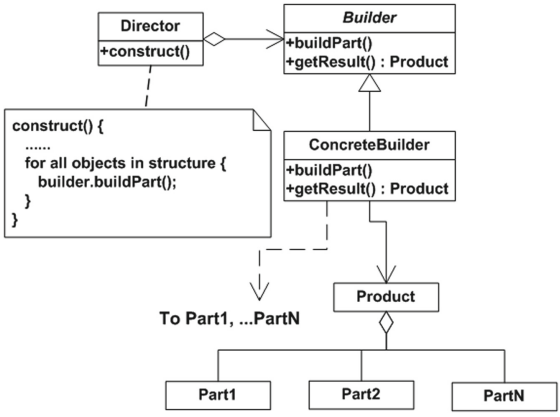

Fig. 5. Builder structure, as defined in the GoF catalog.

Figure 4, gives an example of Composite from Source Making. `AbstractFile`, `File` and `Directory`, correspond to `Component`, `Leaf` and `Composite`. `Directory` is an incomplete participant because it does not provide methods for the removal and the retrieval of `AbstractFile` objects.

4.4 Erroneous Participants

In the corpus, we also found **erroneous participants** that do not behave as dictated in the original definitions of the patterns. Typically, the deviations of the participants' behaviors are such that they jeopardize the intent of the pattern. We observed erroneous participants only in the pattern examples. Erroneous participants have a direct impact on the correctness of the examples in which they appear. In all cases, the examples are wrong.

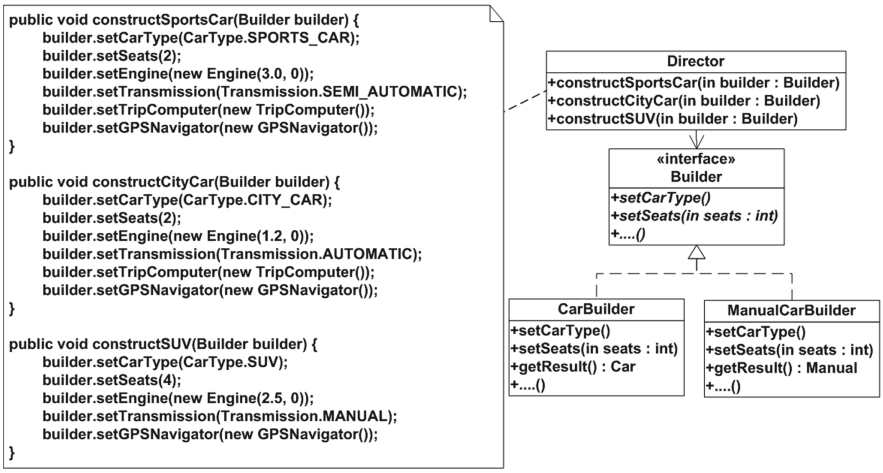

Fig. 6. Example of Builder from Refactoring Guru.

Figure 5 gives the original structure of Builder. The intent of this pattern is to *"separate the construction of a complex object from its representation so that the same construction process can create different representations"*. Consequently, the same construction process can be reused to create objects with different internal representations. `Builder`, defines an interface that provides operations for the creation of the constituents parts of a `Product` object. `ConcreteBuilder`, is an implementation of the `Builder` interface that constructs and assembles the parts of the `Product` object. In general, `Builder` can have different alternative implementations that correspond to different internal `Product` object representations. `ConcreteBuilder` further provides a method for retrieving the resulting `Product` object. `Director`, realizes the overall `Product` object construction process, by invoking methods of the `Builder` interface.

Figure 6 details an example of Builder from Refactoring Guru. In the example, the `Builder` interface has two alternative implementations, namely, `CarBuilder` and `ManualCarBuilder`. The products of `CarBuilder` and `ManualCarBuilder` are `Car` and `Manual` objects, respectively. The constituent parts of `Car` and `Manual` objects are `Engine`, `Transmission`, `TripComputer` and `GPSNavigator` objects. `Director` realizes the object construction process. However, the `Director` class is not correct with respect to the pattern specification. The main problem is that the `Director` class implements three different construction processes, instead of one. The different construction processes are similar, in fact they are code clones, and depend on the internal representation of the objects under construction.

Table 3. Assessing the compliance of pattern definitions.

Deviations in pattern definitions				
Patterns	**Source Making**	**Refactoring Guru**	**Tutorials Point**	**Java T Point**
Abstract Factory	0	0	1	0
Builder	0	0	1	0
Factory Method	0	0	0	0
Prototype	0	0	0	0
Singleton	0	0	0	0
Adapter	0	0	0	0
Bridge	0	0	0	0
Composite	1	1	0	0
Decorator	0	0	0	0
Façade	0	0	0	0
Flyweight	1	2	0	0
Proxy	0	0	0	0
Chain of Resp	1	0	0	0
Command	1	0	0	0
Interpreter	0		0	0
Iterator	0	2	0	0
Mediator	2	1	0	0
Memento	0	0	0	0
Observer	1	1	0	1
State	0	0	0	0
Strategy	0	0	0	0
Template Method	0	0	0	0
Visitor	1	0	1	
Sum	8	7	3	1
Density of deviations in pattern definitions	0.35	0.32	0.13	0.05
% Patterns with deviating definitions	30.43%	27.27%	13.04%	9.09%

The two classes that implement the `Builder` interface are also incorrect. In particular, the two classes do not construct the parts of the resulting objects, as dictated by the pattern. Instead, `Director` constructs the parts and gives them to the `Builder` implementation classes as parameters of respective methods. Consequently, `Director` is not independent from the internal representation of the objects under construction. Overall, the example fails to communicate the intent of the Builder pattern.

5 Compliance of Pattern Definitions and Examples

In this section, we assess the compliance of pattern definitions and examples to the original GoF pattern definitions.

We begin our assessment from the **pattern definitions** that we consider in this study. Table 3, summarizes the results of the assessment. Specifically, for each site the table provides (1) the number of deviations that we observed in the definition of each pattern, and in total, (2) the density of deviations in the pattern definitions, defined as the total number of deviations in pattern definitions, divided by the number of pattern definitions, and (3) the percentage of patterns with deviating definitions. The empty cells in the table concern patterns for which there are no definitions in the respective sites.

Overall, we observe that **patterns with deviating definitions do not occur very often**. In the sites that we examined, the percentage of patterns with deviating definitions varies from 9.09% to 30.43%. The **density of deviations in the pattern definitions is low**, ranging from 0.05 deviations per definition to 0.35 deviations per definition. In practice, this means that most definitions adhere to the original GoF definitions. In Source Making we observe the highest density of deviations, followed by Refactoring Guru, Tutorials Point and Java T Point. In all sites, there are eleven patterns with deviation-free definitions.

Next, we investigate the compliance of the **pattern definitions** that we consider in our study, in relation to the **different kinds of deviations** that occur in the definitions. To this end, for each site, Table 4 gives the number of deviations of each kind that occur in the definitions, and the density of the deviations of each kind.

Table 4. Compliance of pattern definitions for the different kinds of deviations.

Kinds of deviations in pattern definitions								
	Source Making		Refactoring Guru		Tutorials Point		Java T Point	
	# deviations	density	# deviations	density	# deviations	density	# deviations	density
Intent Deviations	0	0.00	0	0.00	3	0.13	1	0.05
Missing Participant	7	0.30	4	0.18				
Incomplete Participant	1	0.04	3	0.14				

In the results, we observe a **low density of intent deviations** in the patterns definitions. In Source Making and Refactoring Guru, there are no intent deviations. In Source Making and Refactoring Guru, **the density of missing participants is higher than the density of incomplete participants**. The cells that concern missing and incomplete participants in Tutorials Point and Java T Point are empty because the pattern definitions in these sites are very brief, consisting only of the intent of the patterns. The structure of the patterns is not part of the provided definitions. The brevity of the definitions is also

the reason for the relatively small number of deviations and the respective low deviation density values in Tutorials Point and Java T Point. Despite the low deviation density values, in these two sites we observe the only occurrences of intent deviations, which result in entirely incorrect pattern definitions. Moreover, 3 of the 11 occurrences of missing participants also result in incorrect pattern definitions that do not reflect the original purpose of the patterns. The rest of the deviations, result in incomplete definitions that partially specify the structure of the original GoF patterns.

Table 5. Assessing the compliance of pattern examples.

Deviations in pattern examples								
	Source Making		Refactoring Guru		Tutorials Point		Java T Point	
Patterns	# deviations	# examples	# deviations	# examples	# deviations	# examples	# deviations	# examples
Abstract Factory	12	7	0	10	2	1	3	1
Builder	1	5	1	10	3	1	3	1
Factory Method	5	6	1	10	1	1	1	1
Prototype	5	7	7	10	0	1	1	1
Singleton	1	4	0	10	0	1	0	1
Adapter	8	6	7	10	1	1	1	1
Bridge	0	5	0	10	0	1	0	1
Composite	19	9	20	10	4	1	3	1
Decorator	2	8	1	10	0	1	0	1
Façade	0	5	0	10	0	1	0	1
Flyweight	13	7	20	10	1	1	0	1
Proxy	3	6	0	10	0	1	0	1
Chain of Resp	10	6	0	10	1	1	1	1
Command	9	7	1	10	0	1	0	1
Interpreter	8	5			0	1	3	1
Iterator	17	6	23	10	2	1	2	1
Mediator	15	6	0	10	3	1	2	1
Memento	3	5	11	10	1	1	1	1
Observer	16	8	12	10	2	1	1	1
State	4	9	0	10	0	1	2	1
Strategy	3	5	0	10	0	1	0	1
Template Method	0	5	0	10	0	1	0	1
Visitor	0	6	0	10	0	1		
Sum	154	143	104	220	21	23	24	22
Density of deviations in pattern examples	1.08		0.47		0.91		1.09	
% Patterns with deviating examples	86.36%		52.17%		50.00%		63.64%	

We move on to the assessment of the **pattern examples** that we consider in our study. The results of the assessment are given in Table 5. In particular, for each site the table reports (1) the number of examples for each pattern, (2) the number of deviations that we observed in the examples, (3) the density of deviations in the examples, defined as the total number of deviations, divided by the number of pattern examples, and (4) the percentage of patterns with deviating examples. The empty cells in the table signify the lack of pattern examples in the corresponding sites.

In the results, we observe that **patterns with deviating examples are quite frequent**. Specifically the percentage of patterns with deviating examples

Table 6. Compliance of pattern examples for the different kinds of deviations.

Kinds of deviations in pattern examples								
	Source Making		Refactoring Guru		Tutorials Point		Java T Point	
	# deviations	density	# deviations	density	# deviations	density	# deviations	density
Missing Participant	108	0.76	49	0.22	14	0.61	15	0.68
Incomplete Participant	28	0.20	45	0.20	4	0.17	5	0.23
Errorneous Participant	17	0.12	10	0.05	3	0.13	6	0.27

in the examined sites ranges from 50% to 86.6%. The **density of deviating examples is medium-high**, varying from 0.47 to 1.09 deviations per example. In all sites, there are only **three patterns** with **deviation-free examples**. In three out of the four sites, the density of deviations in the examples is greater than 0.9. Practically this means that **in many pattern examples we have multiple deviations**. Among the sites, Source Making is the one with the highest density of deviations, followed by Java T Point, Refactoring Guru and Tutorials Point. In all sites, **the percentage of patterns with deviating examples is higher than the percentage of patterns with deviating definitions**.

Regarding the **different kinds of deviations**, Table 6 gives the number of deviations of each kind that occur in the examples, and the density of deviations of each kind. Among the different kinds of deviations, **missing participants are the ones that occur more often, followed by incomplete participants and erroneous participants**. In all sites, **the number and the density of missing participants is high**. On the other hand, **the numbers and the densities of incomplete and erroneous participants are low**. Overall, 60 of the 186 occurrences of missing participants and 20 of the 82 occurrences of incomplete participants, result in incorrect examples. The same holds for all the occurrences of erroneous participants. The rest of the deviations, result in examples that partially illustrate the structure of the original GoF patterns.

Threats to Validity: The retrieval of the examined definitions and examples has been done manually. The identification of deviations and the compliance assessment of the retrieved pattern definitions and examples has also been done manually. This is a possible threat to the construct validity of the study. To mitigate this risk and reduce the probability of human mistakes the gathering and the assessment of the data have been done in multiple iterations. Internal validity, is not an issue in our study because we do not attempt to establish any particular cause-effect relationships regarding the deviations that occur in the examined pattern definitions and examples. Regarding external validity, the scope of our study is pattern definitions and examples that we find on the Web. In this context, we studied pattern definitions and examples gathered from four well-known sites. Therefore, we are confident that our findings are representative of the scope of the study.

6 Takeaway Messages for the Developers

In this paper, we assessed the compliance of the pattern definitions and examples that we find on the Web, to the original GoF pattern definitions. Our study brought out the following key messages for the developers:

- The developers should know that the definitions and examples that we find on the Web deviate from the original definitions.
- The pattern definitions that we find on the Web may involve intent deviations, missing participants and incomplete participants, while the pattern examples may involve missing participants, incomplete participants and erroneous participants.
- The impact of the different kinds of deviations to the correctness of the pattern definition and examples varies. Intent deviations and erroneous participants always result in incorrect definitions and examples. Missing participants and incomplete participants may also result in incorrect pattern definitions and examples. However, most of these deviations do not entirely jeopardize the involved pattern definitions and examples. Typically, these deviations result in definitions and examples that partially illustrate the original pattern concepts.
- The developers should be more concerned about deviating examples than deviating definitions, since the frequency of the former is much higher than the frequency of the latter.
- Finally, the developers should be aware that the choice of the site in which they seek information about GoF patterns is important. Certain sites appear more suitable for developers who are looking for pattern definitions and examples that fully comply to the original GoF pattern definitions, while other sites may be more appropriate for developers looking for simplified pattern variants. In any case, the developers should be very careful and crosscheck the information they find on the Web with the original Gof pattern information.

Besides the aforementioned takeaway messages, our study opens up a number of directions for future research. Specifically, a more detailed analysis of the specific deviations that occur in the definitions and the examples of each pattern would be interesting for the developers who seek information about specific patterns. Additional studies that involve the assessment of further sites and pattern collections would also be interesting. Finally, another issue worth investigating in the future is the (semi)automated validation of pattern definitions and examples that we find on the Web.

References

1. Alfadel, M., Aljasser, K., Alshayeb, M.: Empirical study of the relationship between design patterns and code smells. PLoS ONE **15**, e0231731 (2020)
2. Almadi, S.H.S., Hooshyar, D., Ahmad, R.B.: Bad smells of gang of four design patterns: a decade systematic literature review. Sustainability **13**, 10256 (2021)

3. Aversano, L., Canfora, G., Cerulo, L., Grosso, C.D., Penta, M.D.: An empirical study on the evolution of design patterns. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC-FSE), pp. 385–394. ACM (2007)
4. Bieman, J.M., Straw, G., Wang, H., Munger, P.W., Alexander, R.T.: Design patterns and change proneness: an examination of five evolving systems. In: Proceedings of the 9th IEEE International Software Metrics Symposium (METRICS), pp. 40–49 (2003)
5. Dale, M.R., Izurieta, C.: Impacts of design pattern decay on system quality. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement ESEM, pp. 37:1–37:4 (2014)
6. Feitosa, D., Ampatzoglou, A., Avgeriou, P., Nakagawa, E.Y.: Correlating pattern grime and quality attributes. IEEE Access **6**, 23065–23078 (2018)
7. Data are available at https://drive.google.com/file/d/1vAn58ul7whaXM01TMFcj1er5UcCSzrYN/view?usp=sharing
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
9. Izurieta, C., Bieman, J.M.: A multiple case study of design pattern decay, grime, and rot in evolving software systems. Softw. Qual. J. **21**(2), 289–323 (2013). https://doi.org/10.1007/s11219-012-9175-x
10. Mayvan, B.B., Rasoolzadegan, A., Yazdi, Z.G.: The state of the art on design patterns: a systematic mapping of the literature. J. Syst. Softw. **125**, 93–118 (2017)
11. Prechelt, L., Unger, B., Philippsen, M., Tichy, W.F.: Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. IEEE Trans. Softw. Eng. **28**(6), 595–606 (2002)
12. Prechelt, L., Unger, B., Tichy, W.F., Brössler, P., Votta, L.G.: A controlled experiment in maintenance comparing design patterns to simpler solutions. IEEE Trans. Softw. Eng. **27**(12), 1134–1144 (2001)
13. Reimanis, D., Izurieta, C.: Behavioral evolution of design patterns: understanding software reuse through the evolution of pattern behavior. In: Peng, X., Ampatzoglou, A., Bhowmik, T. (eds.) ICSR 2019. LNCS, vol. 11602, pp. 77–93. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22888-0_6
14. Vokác, M.: Defect frequency and design patterns: an empirical study of industrial code. IEEE Trans. Softw. Eng. **30**(12), 904–917 (2004)
15. Walter, B., Alkhaeir, T.: The relationship between design patterns and code smells: an exploratory study. Inf. Softw. Technol. **74**, 127–142 (2016)
16. Zarras, A.: The strategy configuration problem and how to solve it. In: Proceedings of the ACM European Conference on Pattern Languages of Programs (EuroPLoP), pp. 9:1–9:11. ACM (2021)
17. Zarras, A.V.: Common mistakes when using the command pattern and how to avoid them. In: Proceedings of the ACM European Conference on Pattern Languages of Programs (EuroPLoP), pp. 4:1–4:9. ACM (2020)