Do Design Patterns Impact Software Quality Positively?

Conference Paper *in* Proceedings of the Euromicro Conference on Software Maintenance and Reengineering, CSMR · May 2008

DOI: 10.1109/CSMR.2008.4493325 · Source: IEEE Xplore

Yann-Gaël Guéhéneuc
Concordia University Montreal

Do Design Patterns Impact Software Quality Positively?

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

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

Abstract

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

1. Introduction

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

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

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

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

We present detailed results for three design patterns: Abstract Factory, Composite, Flyweight and three quality attributes: reusability, understandability, and expandability. Results for other patterns and quality attributes can be found in [5]. We show that, contrary to popular beliefs, patterns *in practice* do not always improve quality attributes, thus providing evidence against common lore. We attempt to explain these results using object-oriented best practices. We conclude on the need for further studies and that patterns should be used with caution because they may actually impede maintenance and evolution.

Section 2 presents related work and their limitations. Section 3 states the hypothesis and objective of the study and presents our data collection and processing. Section 4 describes our quantification method and presents the results of our survey. Section 5 contains a discussion of the results. Section 6 concludes our research, discusses the threats to the validity of our study and introduces future work.

2. Related Work

Since their introduction by Gamma et al. [2] in 1994, there has been a growing interest on the use of design patterns. We present here some lines of work on the impact of patterns on quality.

Lange and Nakamura demonstrated [6] that design patterns can serve as guide in program exploration and thus make the process of program understanding more efficient. However this study was limited to a single quality attribute and to a little number of patterns.

Wydaeghe et al. [12] presented a study on the concrete use of six design patterns. They discussed the impact of these patterns on reusability, modularity,

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

flexibility, and understandability, and the difficulty to concretely implement these patterns. They concluded that not all patterns have a positive impact on quality attributes. Yet, this study was limited to the authors' own experience and it can hardly been generalized to other contexts of development.

Wendorff [11] evaluated the use of design patterns in large commercial systems and concluded that patterns do not necessarily improve their design. Indeed, a design can be over-engineered [4] and the cost of removing patterns is high. He did not perform a study on the impact of patterns on quality and provided only qualitative arguments.

3. Objective and Data Collection

The hypothesis of this study is that design patterns impact quality positively. Our objective is to quantify and qualify this impact on the overall quality of systems to confirm or refute the hypothesis.

We chose to carry an empirical study using a questionnaire because development and maintenance are manual activities performed by engineers. Thus, the engineers' evaluation is important. The reported results are representative of the engineers' experience and provide an accurate evaluation of the impact of patterns on the quality of systems.

3.1. Definition of the Questionnaire

Following our previous work [3] and the work done in [1, 2], we chose the following set of quality attributes, based on their relevance to design patterns.

- Attributes related to design:
  - **Expandability:** The degree to which the design of a system can be extended.
  - **Simplicity:** The degree to which the design of a system can be understood easily.
  - **Reusability:** The degree to which a piece of design can be reused in another design.
- Attributes related to implementation:
  - **Learnability:** The degree to which the code source of a system is easy to learn.
  - **Understandability:** The degree to which the code source can be understood easily.
  - **Modularity:** The degree to which the implementation of the functions of a system are independent from one another.
- Attributes related to runtime:
  - **Generality:** The degree to which a system provides a wide range of functions at runtime.
  - **Modularity at runtime:** The degree to which the functions of a system are independent from one another at runtime.
  - **Scalability:** The degree to which the system can cope with large amount of data and computation at runtime.
  - **Robustness:** The degree to which a system continues to function properly under abnormal conditions or circumstances.

Each quality attribute was evaluated using a six-point Likert scale: `A - Very positive`, `B - Positive`, `C - Not significant`, `D - Negative`, `E - Very Negative`, and `F - Not applicable`. The sixth value allowed respondents not to answer a question if they did not know or were not sure about the impact of a design pattern on a quality attribute.

For every design pattern in [2] and for every quality attribute from our set, the respondents were asked to assess the impact of the pattern on the quality of a system in which the pattern would be used appropriately, as they would during a technical review [9] or possibly while performing a program comprehension-related activity during maintenance and evolution.

The questionnaire is available on the Internet at `http://www.ptidej.net/downloads/`.

3.2. Data Collection and Processing

We collected respondents' evaluations during the period of January to April 2007 by posting our questionnaire on three mailing lists, *refactoring*, *patterns-discussion*, and *gang-of-4-patterns*.

Among the many answers that we received, we selected the questionnaires of 20 software engineers with a verifiable experience in the use of design patterns in software development and maintenance.

This number of collected evaluations is larger than in previous work. Due to the variations between answers, we felt that the differences between `Positive` and `Very Positive` answers were due to some respondents being less strict than others and thus, that their `Very Positive` evaluations were not directly relevant. This fact has been confirmed in discussions with the respondents. For example, for Builder and expandability, we had 19% of respondents considering the pattern `Very Positive` while 63% considered it `Positive` and 18% considered it `Neutral`. Therefore, we chose to aggregate answers `A` and `B` and answers `D` and `E`: `Positive = A and B`, `Neutral = C`, and `Negative = D and E`.

Using the previous three-point Likert scale, we computed the frequencies of the answers on each quality attribute: `Positive`, `Neutral`, and `Negative` and we carried out a Null hypothesis test to assess the perceived impact of the patterns on the quality attributes.

Answers `F` were not considered because they represented situations where the respondents did not know or did not want to evaluate the impact.

Attributes	Composite		A.Factory		Flyweight	
	E	R(%)	E	R(%)	E	R(%)
Expendability	+	0.00	+	0.00	−	1.76
Simplicity	+	5.92	+	30.36	−	0.00
Reusability	+	15.09	+	50.00	−	15.09
Learnability	+	1.76	−	15.09	−	0.00
Understandability	+	5.92	−	15.09	−	0.00
Modularity	+	5.92	+	0.37	−	5.92
Generality	+	1.76	+	1.76	−	0.15
Mod. at Runtime	+	30.36	−	30.36	−	0.15
Scalability	−	30.36	−	1.76	+	1.76
Robustness	−	0.15	−	0.00	−	1.76
	8 + / 2 −		5 + / 5 −		1 + / 9 −	

Table 1. Estimation of the impact of the three design patterns on quality attributes.

4. Analyses

We now present the detailed results of a quantitative and a qualitative analyses for three design patterns: Abstract Factory, Composite, and Flyweight, and the three quality attributes mentioned by the GoF [2, page xiii]: reusability, expandability, and understandability.

We choose the following three design patterns to illustrate our respondents' assessments first because of their popularity—they are among the most commonly used patterns and thus we felt that their evaluation would be more accurate—and second because they appear to be considered as globally positive, globally neutral, and globally negative.

4.1. Quantitative Analysis

Using the results collected from the questionnaires and presented in details elsewhere [5], we carried out Null hypothesis tests to quantify the impact of the design patterns on the quality attributes and then confirm or refute the hypothesis that *design patterns impact software quality positively.*

The Null hypothesis test yields the results summarized in Tables 1, 2. Full results for all the patterns from [2] and details on the test can be found in [5]. In these tables, the sign + means that, with our Null hypothesis test, the impact of the pattern on the quality attribute is positive else the sign is − (it can be negative or neutral). The number next to a sign represents the risk of making this decision.

We computed this risk using the cumulative density of the Bernoulli distribution. All the details are available in [5].

4.2. Qualitative Analysis

Composite. By analysing Table 1, it appears that the Composite pattern is mostly perceived as having a positive impact on the quality of systems. All quality attributes are impacted positively but for scalability and robustness, which are consider neutral. Given the purpose of the Composite pattern, having a neutral impact on scalability is rather surprising.

Abstract Factory. Table 1 shows that half the quality attributes is considered as positively impacted while the other half is not. It is not surprising that the pattern is overall judged as neutral given its purpose and complexity. It is striking that learnability and understandability are felt negatively impacted.

Flyweight. Table 1 reports that this pattern is perceived as impacting negatively all quality attributes but scalability. Given the purpose of the pattern, it is not surprising that its impact on scalability is judged positively. The negative perception could be explained by the less frequent use of Flyweight in comparison with Composite and Abstract Factory.

We choose the following three quality attributes because it is claimed in [2, 10] that they are improved by the use of design patterns.

Expandability. Table 2 presents the analysis of the respondents' evaluations of the impact of the design patterns on expendability. All respondents felt that expandability is improved when using patterns, in conformance with the claims made in [2].

Reusability. Table 2 shows that reusability is felt as being slightly more negatively impacted by design patterns, with 13 neutral or negative patterns and 10 positive patterns. This is rather surprising as the use of patterns is claimed to improve reusability.

Understandability. Table 2 presents the analysis of understandability. Similarly to reusability, respondents felt that understandability was rather slightly negatively impacted by the use of patterns.

5. Discussion of the Results

The analysis of the results of our study reveal that, in contrary to common lore, design patterns do not always impact quality attributes positively. Our respondents consider that, although patterns are useful to solve design problems, they do not always improve the quality of the systems in which they are applied. In particular, a large number of respondents considered that they sensibly decrease simplicity, learnability, and understandability. Some patterns, like Flyweight, are considered as impacting most attributes negatively.

Design Patterns	Expendability(%)		Understandability(%)		Reusability(%)	
	E	R(%)	E	R(%)	E	R(%)
A.Factory	+	0.00	−	15.09	+	50.00
Builder	+	0.15	+	0.37	−	15.09
F.Method	+	1.76	−	30.36	+	15.09
Prototype	+	30.36	+	30.36	+	30.36
Singleton	−	0.15	+	0.15	−	0.37
Adapter	+	30.36	−	30.36	+	5.92
Bridge	+	0.37	+	50.00	−	30.36
Composite	+	0.00	+	5.92	+	15.09
Decorator	+	0.15	−	30.36	−	5.92
Facade	+	30.36	+	1.76	−	5.92
Flyweight	−	1.76	−	0.00	−	15.09
Proxy	−	30.36	−	5.92	+	50.00
Ch.Of.Resp	+	0.15	−	5.92	+	30.36
Command	+	5.92	−	5.92	−	5.92
Interpreter	+	5.92	+	5.92	+	30.36
Iterator	+	0.15	+	50.00	+	5.92
Mediator	+	30.36	+	30.36	−	1.76
Memento	−	5.92	−	30.36	−	15.09
Observer	+	0.15	−	30.36	+	50.00
State	+	5.92	+	30.36	−	1.76
Strategy	+	1.76	+	15.09	−	30.36
T.Method	+	0.37	−	15.09	+	30.36
Visitor	+	5.92	−	1.76	−	1.76
	19 + / 4 −		11 + / 12 −		11 + / 12 −	

Table 2. Estimation of the impact of design patterns on the three quality attributes

We now attempt to explain these results by studying design patterns from the point of view of object-oriented software practices. We focus on some "famous" patterns as we consider their evaluations by the respondents more accurate. By "famous", we mean that all selected respondents fill the entire evaluations of these patterns.

We make the hypothesis that the principles help in improving the quality and thus should explain the results found on the impact of design patterns on quality.

We discuss here the results of the three design patterns, shown in Tables 1 and 2, with respect to object-oriented principles presented in [7].

Discussion for the other patterns from [2] are presented in [5].

Composite. The Composite pattern allows an instance of a class to be treated in the same way as a group of objects. It makes it easy to add new kinds of objects. It makes clients simpler, because they do not have to know if they are dealing with a leaf or a composite object. Thus its use in a system impacts positively the expandability and the simplicity, which is in accordance with the evaluations of our respondents. However, the Composite pattern makes it harder to restrict the type of objects in a composite and may lead to large amount of objects being instantiated and referenced, thus possibly explaining the neutral evaluations of its impact on robustness and scalability.

Abstract Factory. The intent of the Abstract Factory pattern is to separate the creation of objects from their uses. It allows for new derived types to be introduced with no change to the code that uses the base objects. This pattern thus respects the Open Close Principle and improves the expandability, the simplicity, and the generality of systems. Also, it makes it possible to interchange concrete classes without changing the code that uses them, even at runtime, thus improving modularity and reusability, in accordance with the our respondents' evaluations.

However, due to the flexibility of interchanging concrete classes at runtime, the pattern should improve the modularity at runtime and the scalability of systems in which it is used, a position which contradicts our respondents' results. We believe that this unexpected results can be explain by the difficulty of writing optimal implementations of this pattern. The use of this pattern, as with similar design patterns, induces the risk of unnecessary complexity and extra work in the initial design and implementation, thus decreasing understandability and learnability.

Flyweight. The Flyweight pattern is considered by our respondents as impacting negatively most quality attributes. The Flyweight pattern is tied to a very specific problem and thus is not expandable. Yet, it allows thousands of objects to work together improving thus scalability. It is not simple and not generalizable, it decreases learnability, understandability, and reusability, as software engineers must know the specific solved problem to be able to understand the implementation. This pattern violates the Open Close Principle as engineers cannot extend the piece of code in which it is used without almost rewriting it all.

From this study and an extended study [5], we remark that most design patterns respect the principles of object-oriented programming.

Hence, according to our hypothesis that object-oriented best practices help producing systems with good quality, it is surprising that their use seems to decrease quality. A possible explanation could be that, for a pattern, many implementations are possible and that the concrete implementations may not be conform to the principles of object-oriented programming. Among the 23 patterns from the GOF, some are not frequently used in systems. Thus, the negative evaluations may be just an *a priori* on the pattern because our respondents considered the pattern to be not suitable and then their evaluation may not reflect the real impact of the implementation of the pattern on quality.

It may also be that these best practices are necessary but not sufficient to build systems with good quality. In addition, we notice that, for the studied design patterns, several principles do not seem to apply or explain the results of the study, thus calling for further studies on the impact of these principles on quality.

6. Conclusion and Threats to Validity

With this study, we show that design patterns do not always improve the quality of systems. Some patterns are reported to decrease some quality attributes and to not necessarily promote reusability, expandability, and understandability. Therefore, we bring further evidence that design patterns should be used with caution during development because they may actually impede maintenance and evolution. This study also reveals that object-oriented principles may not be so "good" as they may not necessarily result in systems with good quality. Thus, there is a need for studies to assess the impact of these principles on the quality of systems.

There is no threat to the validity of the conclusion of this study as there is a direct relationship between the design of a system and its quality. The design of a system directly impacts the quality attributes presented in Section 3.1 thus we can say that there is no threat to the construct and internal validities of our study. However, the results of our study may not be fully generalisable to any software engineers, patterns, and systems. For future work we plan to continue collecting evaluations to improve the accuracy of our results and to generalise our conclusions to different software context. The questionnaire is available on the Internet at `http://www.ptidej.net/downloads/` (it may take some minutes to load as it weighs 4 MB). We are looking forward receiving more evaluations.

Acknowledgments

We are grateful to Kim Mens for the fruitful discussions. We would like to thank all the respondents for their evaluations and comments. This work has been partially funded by NSERC and the VINCI program of University of Montreal.

References

[1] J. Bansiya and C. G. Davis. A hierarchical model for object-oriented design quality assessment. In *IEEE Transactions on Software Engineering*, 28:4–17, January 2002.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1st edition, 1994.

[3] Y.-G. Guéhéneuc, J.-Y. Guyomarc'h, K. Khosravi, and H. Sahraoui. Design patterns as laws of quality. University of Montreal, 2005.

[4] J. Kerievsky. *Refactoring to Patterns*. Addison-Wesley, 1st edition, August 2004.

[5] F. Khomh and Y.-G. Guéhéneuc. An empirical study of design patterns and software quality. Technical Report 1315, University of Montréal, january 2008. `http://www.iro.umontreal.ca/~ptidej/Publications/~Documents/Research+report+DP+Quality+January08.doc.pdf`.

[6] D. B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications*, pages 342 – 357. ACM Press, 1995.

[7] R. C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. 2002.

[8] W. B. McNatt and J. M. Bieman. Coupling of design patterns: Common practices and their benefits. In *Proceedings of the 25th Computer Software and Applications Conference*, pages 574–579. IEEE Computer Society Press, October 2001.

[9] R. S. Pressman. *Software Engineering – A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, November 2001.

[10] B. Venners. How to use design patterns – A conversation with Erich Gamma, part I, May 2005. `http://www.artima.com/lejava/articles/gammadp.html`.

[11] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *Proceedings of 5th Conference on Software Maintenance and Reengineering*, pages 77–84. IEEE Computer Society Press, March 2001.

[12] B. Wydaeghe, K. Verschaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers. Building an OMT-editor using design patterns: An experience report. 1998.