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

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

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

**Abstract.** Although the belief of utilizing design patterns to create better quality software is fairly widespread, there is relatively little research objectively indicating that their usage is indeed beneficial.

In this paper we try to reveal the connection between design patterns and software maintainability. We analyzed more than 300 revisions of JHotDraw, a Java GUI framework whose design relies heavily on some well-known design patterns. We used our probabilistic quality model for estimating the maintainability and we parsed the javadoc annotations of the source code for gathering the pattern instances.

We found that every introduced pattern instance caused an improvement in the different quality attributes. Moreover, the average design pattern line density showed a very high, 0.89 Pearson correlation with the estimated maintainability values. Although the amount of available empirical data is still very small, these first results suggest that the usage of design patterns do improve code maintainability.

**Keywords:** Design patterns, Software maintainability, Empirical validation, OO design.

## 1 Introduction

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

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

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

T.-h. Kim et al. (Eds.): ASEA/DRBC 2012, CCIS 340, pp. 138–145, 2012.

approach for the empirical evaluation. To get absolute measure for the maintainability of a system we used our probabilistic quality model [2]. Our subject system was JHotDraw 7, a Java GUI framework for technical and structured Graphics (`http://www.jhotdraw.org/`). Its design relies heavily on some well-known design patterns. Instead of using different design pattern mining tools we parsed the javadoc entries of the system directly to get all the applied design patterns. We analyzed more than 300 revisions of JHotDraw, calculated the maintainability values and mined the design pattern instances. We gathered this empirical data with the following research questions in mind:

**Research Question 1.** *Are there any traceable impacts of the application of design patterns on software maintainability?*

**Research Question 2.** *What kind of relation exists between the design pattern density and the maintainability of the software?*

There are some promising results showing that applying design patterns improve the different quality attributes according to our maintainability model. In addition, the ratio of the source code lines taking part in some design patterns in the system has a very high correlation with the maintainability in case of JHotDraw. However, these results are only a small step towards the empirical analysis of design patterns and software quality.

The rest of our paper is structured as follows. In Section 2 we highlight the related work then in Section 3 we present our approach for analyzing the relationship between design patterns and maintainability. Section 4 summarizes the achieved empirical results. Next, Section 5 lists the possible threats to the validity of our work. Finally, we conclude the paper in Section 6.

## 2 Related Work

Although the concept of utilizing design patterns in order to create better quality software is fairly widespread, there is relatively little research that would objectively indicate that their usage is indeed beneficial.

Since design patterns and software metrics are both geared towards the same goal - improving quality - Huston [10] attempts to prove their correlation by representing the system's classes in connection matrices and defining algorithms for applying patterns and evaluating metrics. This approach shows promising results but it is purely theoretical.

In an empirical study, - replicated in 2004 [18] and in 2011 [12] - Prechelt et al. [15] gave groups identical maintenance tasks to perform on two different versions - with and without design patterns - of four programs. Here, the impact on maintainability was measured by completion time and correctness while this article uses objective quality metrics and analyzes a more complex software.

In another case study, Vokáč [17] measured the defect frequency of pattern classes versus other classes in an industrial C++ source for three years and concluded that some patterns - Singleton, Observer - tend to indicate more complex parts than others, e.g.: Factory. However, the used pattern mining method could

have introduced false positives or true negatives and the defects are also based on subjective reports. In contrast, we rely on the official pattern documentation of source code and the quality model published in [2].

Khomh and Guéhéneuc [11] used questionnaires to collect the opinions of 20 experts on how each design pattern helps or hinders them during maintenance. They bring evidence that design patterns should be used with caution during development because they may actually impede maintenance and evolution. Another experiment conducted by Ng et al. [14] decomposes maintenance tasks to subtasks and examines the frequency of their use according to the deployed design patterns and whether these patterns are utilized during the change. They statistically conclude that performing whichever task while taking existing patterns into consideration yields less faulty code. Trying to evaluate the effectiveness of patterns in software evolution, Hsueh et al. [9] defined their context and their anticipated changes and then checked whether they held up to the expectations. Their conclusion is that although design patterns can be misused, they are effective to some degree in either short or long term maintenance. Aversano et al. [1] also investigate pattern evolution by tracking their modifications and how many other, possibly unrelated modifications they cause. In this paper we do not use questionnaires or evaluate design patterns manually, but rather measure its impact on maintainability directly. Moreover, we focus on their impact on the maintainability of the system as a whole and not only the evolution of the code implementing design patterns.

## 3 Approach

For analyzing the relationship between design patterns and maintainability we calculate the following measures for every revision of JHotDraw system:

- $M_r$ - an absolute measure of maintainability for the revision $r$ of the system. We used our probabilistic quality model [2] to get this absolute measure.
- $TLLOC$ - the total number of logical lines of code in the system (computed by the Columbus toolset [6]).
- $TNCL$ - the total number of classes in the system.
- $PIn_r$ - the number of pattern instances in revision $r$ of the system.
- $PCl_r$ - the number of classes playing a role in any pattern instances in revision $r$ of the system.
- $PLn_r$ - the total number of logical lines of classes playing a role in any pattern instances in revision $r$ of the system.
- $PDens_r$ - the pattern line density of the system defined by the following formula: $\frac{PLn_r}{TLLOC}$

To answer our research questions we examine the tendency of $M_r$ in comparison to the pattern related metrics and changes in the number of pattern instances. The pattern related metrics are calculated by our own tool that is able to process the structured *javadoc* comments.

### 3.1 Probabilistic Software Quality Model

Our probabilistic software quality model is based on the quality characteristics defined by the ISO/IEC 9126 standard. The computation of the high level quality characteristics is based on a directed acyclic graph whose nodes correspond to quality properties that can either be internal (low-level) or external (high-level). Internal quality properties characterize the software product from an internal (developer) view and are usually estimated by using source code metrics. External quality properties characterize the software product from an external (end user) view and are usually aggregated somehow by using internal and other external quality properties. The edges of the graph represent dependencies between an internal and an external or two external properties. The aim is to evaluate all the external quality properties by performing an aggregation along the edges of the graph, called Attribute Dependency Graph (ADG). We used the particular ADG presented in [2] for assesing the maintainability of JHotDraw Java system.

### 3.2 Mining Design Patterns

Instead of applying one of the design pattern miner tools (e.g. [5, 19]) we used a more direct way for extracting pattern instances from different JHotDraw versions. Since every design pattern instance is documented in JHotDraw 7 we could easily build a text parser application to collect all the patterns. This approach guarantees that no false positive instances are included and no true negative instances are left out from the empirical analysis. Finally we ran the parser on all relevant revisions of JHotDraw7 to track the changes.

## 4 Results

We analyzed all the 779 revisions of the JHotDraw 7 subversion branch[1] and calculated the measures introduced in Section 3. The documentation of design patterns is introduced in revision 522, therefore the empirical evaluation has been performed on 258 revisions (between revision 522 and 779). Some basic properties of the starting and ending revision of the analyzed JHotDraw system can be seen in Table 1.

**Table 1.** Basic properties of JHotDraw 7 system

| **Revision** ($r$) | **Lines of code** | **Nr. of packages** | **Nr. of classes** | **Nr. of methods** | $PIn_r$ | $\frac{PCl_r}{TNCL}$ |
|---|---|---|---|---|---|---|
| 522 | 72472 | 54 | 630 | 6117 | 45 | 11.58% |
| 779 | 81686 | 70 | 685 | 6573 | 54 | 13.28 % |

To be able to answer our first research question we have analyzed those particular revisions where the number of design pattern instances has changed. After

[1] `https://jhotdraw.svn.sourceforge.net/svnroot/jhotdraw/trunk/jhotdraw7/`

filtering out the changes that does not introduce or remove real pattern instances (e.g.: comments are added to an already existing pattern instance) five revisions have remained. We also checked that these change sets do not contain a lot of source code that is not related to patterns. It is important to be able to clearly distinguish the effect of design pattern changes to maintainability. In all five cases more than 90% of the code changes are related to the pattern implementations. The tendency of different quality attributes in these revisions can be seen in Table 2.

**Table 2.** The tendency of the quality attributes in case of design pattern changes

| **Revision** ($r$) | **Pattern** | **Pattern Line Density** ($PDens_r$) | **Maintainability** ($M_r$) | **Testability** | **Analyzability** | **Stability** | **Changeability** |
|---|---|---|---|---|---|---|---|
| 531 | +3 | ↗ | ↗ | ↗ | ↗ | ↗ | ↗ |
| 574 | +1 | ↗ | ↗ | ↗ | ↗ | ↗ | ↗ |
| 609 | –1 | ↘ | — | — | — | — | — |
| 716 | +1 | ↘ | ↗ | ↗ | ↗ | ↗ | ↗ |
| 758 | +1 | ↗ | ↗ | ↗ | ↗ | ↗ | ↗ |

In four out of five cases there was growth in the pattern instance numbers. In all four cases every ISO/IEC 9126 quality characteristic (including the maintainability) increased compared to the previous revision. This is true even for revision 716 where the pattern line ratio decreased despite the addition of a design pattern. In case of revision 609 a *Framework* pattern has been removed but the quality characteristics have remained unchanged. This is not so surprising since this pattern (which is not part of the GoF patterns) consists of a simple interface. Therefore its removal does not have any effect on the low level source code metrics on what our maintainability model is based on.

As we have shown in one of our previous works [3] a system's maintainability does not improve during development without applying explicit refactorings. Therefore, the application of design patterns can be seen as applying refactorings on the source code. These results support the hypothesis that design patterns do have a traceable impact on maintainability. In addition, our empirical analysis on JHotDraw indicates that this impact is positive.

To shed light on the relationship between design pattern density and maintainability we performed a correlation analysis on pattern line density ($PDens_r$) and maintainability ($M_r$). We chose pattern line density instead of pattern instance or pattern class density because it is the finest grained measure showing the amount of source code related to any pattern instances. Figure 1 depicts the tendencies of pattern line density and maintainability. It can be seen that the two curves have a similar shape meaning that they move very much together. The Pearson correlation analysis of the entire data set (from revision 522 to 779) shows the same result, the pattern line density and maintainability has a **0.89** correlation. This result may indicate that there is a strong relation between the rate of design patterns in the source code and the maintainability. However, this is still a surmise and we cannot generalize the results without performing a large number of additional empirical evaluations.

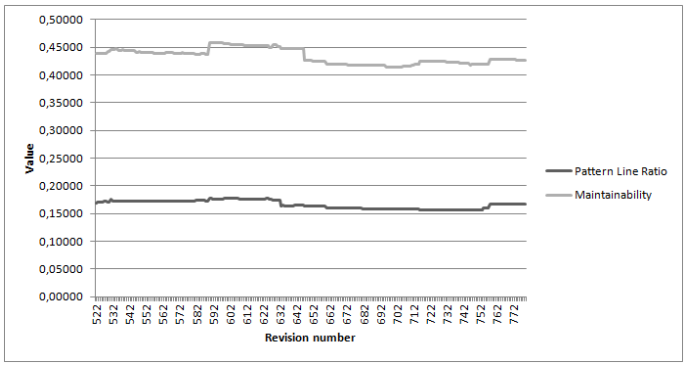


**Fig. 1.** The tendencies of pattern line density and maintainability

## 5 Threats to Validity

Similarly to most of the works, our approach also has some threats to validity. First of all, when dealing with design patterns, the accuracy of mining is always a question. As there are no provably perfect pattern miner tools, we chose our subject system to be a special one, having all design pattern instances documented by its authors. This way we can be sure that all (intentionally placed) design patterns are recognized and no false positive instances are introduced. Of course it is still possible that some pattern comments are missing or our text parser introduces false instances. We reduced this effect by manually inspecting the results of our text parser as well as the source code of JHotDraw.

Another threat to validity is using our previously published quality model for calculating maintainability values. Although we have done some empirical validation on our probabilistic quality model in our previous work, we cannot state that the used maintainability model is fully validated. Moreover, as the ISO/IEC 9126 standard is not defining the low-level metrics, the results can vary depending on the quality model's settings (chosen metrics and weights given by professionals). These factors are possible threats to validity, but our first results and continuous empirical validation of the maintainability model proves its applicability and usefulness.

Finally, the small number of design pattern changes and the fact that less than 300 revisions of one system have been evaluated threatens the generality of results. It might be possible that the explored relationship between design patterns and maintainability is just a byproduct of other factors. Our analysis is only a first step towards the empirical analysis of this relation. Nonetheless, these first results are already valuable and support the common belief that design patterns do have a positive impact on maintainability.

## 6 Conclusions

In this paper we presented an empirical analysis of exploring the connection between design patterns and software maintainability. We analyzed nearly 300

revisions of JHotDraw 7 and calculated the maintainability values with our probabilistic quality model and mined the design pattern instances parsing the comments in the source code. Examining the maintainability values where changes in the number of pattern instances happened and by correlation analysis of design pattern density and maintainability we were able to draw some conclusions.

Every ISO/IEC 9126 quality characteristics (including the maintainability) increased with the number of pattern instances. Since there were no other changes in the code it indicates that the quality attributes increased due to the introduced patterns. Hence, we could observe a traceable positive impact of design patterns to maintainability of the subject system.

Another interesting result is that the pattern line density and maintainability values have a very similar tendency. The Pearson correlation analysis of the data sets showed that there is a strong relation between the rate of design patterns in the source code and its maintainability. These facts strengthen the common assumption that using design patterns improve the maintainability of the source code. However, these results should be handled with caution. We analyzed only one system and a relatively few number of pattern instance changes. We are far from drawing some general conclusions based on this; our work should be considered as a first step towards the empirical validation of the relation between design patterns and software maintainability.

**Acknowledgements.** This research was supported by the Hungarian national grant GOP-1.1.1-11-2011-0049 and the European Union co-funded by the European Social Fund, TÁMOP-4.2.2/B-10/1-2010-0012.

## References

1. Aversano, L., Canfora, G., Cerulo, L., Del Grosso, C., Di Penta, M.: 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 Symposium on the Foundations of Software Engineering, ESEC-FSE 2007, pp. 385–394. ACM, New York (2007)
2. Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R., Gyimóthy, T.: A Probabilistic Software Quality Model. In: Proceedings of the 27th IEEE International Conference on Software Maintenance, ICSM 2011, pp. 368–377. IEEE Computer Society, Williamsburg (2011)
3. Bakota, T., Hegedűs, P., Ladányi, G., Körtvélyesi, P., Ferenc, R., Gyimóthy, T.: A Cost Model Based on Software Maintainability. In: Proceedings of the 28th IEEE International Conference on Software Maintenance, ICSM 2012. IEEE Computer Society, Williamsburg (2012)
4. Bansiya, J., Davis, C.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering 28, 4–17 (2002)
5. Dong, J., Lad, D.S., Zhao, Y.: DP-Miner: Design Pattern Discovery Using Matrix. In: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2007, pp. 371–380. IEEE Computer Society, Washington, DC (2007)

6. Ferenc, R., Beszédes, Á., Tarkiainen, M., Gyimóthy, T.: Columbus – Reverse Engineering Tool and Schema for C++. In: Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002), pp. 172–181. IEEE Computer Society (October 2002)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Pub. Co. (1995)
8. Heitlager, I., Kuipers, T., Visser, J.: A Practical Model for Measuring Maintainability. In: Proceedings of the 6th International Conference on Quality of Information and Communications Technology, pp. 30–39 (2007)
9. Hsueh, N.L., Wen, L.C., Ting, D.H., Chu, W., Chang, C.H., Koong, C.S.: An Approach for Evaluating the Effectiveness of Design Patterns in Software Evolution. In: IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 315–320 (July 2011)
10. Huston, B.: The Effects of Design Pattern Application on Metric Scores. Journal of Systems and Software, 261–269 (2001)
11. Khomh, F., Guéhéneuc, Y.G.: Do Design Patterns Impact Software Quality Positively? In: Proceedings of the 12th European Conference on Software Maintenance and Reengineering, CSMR 2008, pp. 274–278. IEEE Computer Society, Washington, DC (2008)
12. Krein, J., Pratt, L., Swenson, A., MacLean, A., Knutson, C., Eggett, D.: Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University. In: Second International Workshop on Replication in Empirical Software Engineering Research (RESER 2011), pp. 25–34 (September 2011)
13. McNatt, W.B., Bieman, J.M.: Coupling of Design Patterns: Common Practices and Their Benefits. In: Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, COMPSAC 2001, pp. 574–579. IEEE Computer Society, Washington, DC (2001)
14. Ng, T.H., Cheung, S.C., Chan, W.K., Yu, Y.T.: Do Maintainers Utilize Deployed Design Patterns Effectively? In: Proceedings of the 29th International Conference on Software Engineering, ICSE 2007, pp. 168–177. IEEE Computer Society, Washington, DC (2007)
15. Prechelt, L., Unger, B., Tichy, W., Brössler, P., Votta, L.: A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. IEEE Transactions on Software Engineering 27, 1134–1144 (2001)
16. Venners, B.: How to Use Design Patterns - A Conversation With Erich Gamma, Part I (2005), `http://www.artima.com/lejava/articles/gammadp.html`
17. Vokáč, M.: Defect Frequency and Design Patterns: an Empirical Study of Industrial Code. IEEE Transactions on Software Engineering 30(12), 904–917 (2004)
18. Vokáč, M., Tichy, W., Sjøberg, D.I.K., Arisholm, E., Aldrin, M.: A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns - A Replication in a Real Programming Environment. Empirical Software Engineering 9(3), 149–195 (2004)
19. Wendehals, L.: Improving Design Pattern Instance Recognition by Dynamic Analysis. In: Proceedings of the ICSE 2003 Workshop on Dynamic Analysis (WODA), Portland, USA (2003)
20. Wendorff, P.: Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project. In: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, CSMR 2001, p. 77. IEEE Computer Society, Washington, DC (2001)