Original source of publication: Evolution of Design Patterns: a Replication Study, ESEM'14 September 18-19, 2014, Torino, Italy, DOI: http://dx.doi.org/10.1145/2652524.2652563
The original publication is available at http://dl.acm.org/citation.cfm?id=2652563&CFID=691271060&CFTOKEN=52232984

Evolution of Design Patterns: a Replication Study

Bruno Rossi
Department of Computer Systems and Communications,
Masaryk University, Brno, Czech Republic
brossi@mail.muni.cz

Barbara Russo
Faculty of Computer Science,
Free University of Bozen-Bolzano, Bolzano, Italy
brusso@unibz.it

ABSTRACT

Context. In 2007, Aversano et al. [2] analysed the evolution of JHotDraw, ArgoUML, and Eclipse JDT between years 2000-2005 to understand the role of frequently changed design patterns. **Goal.** In this paper, we perform a replication of the study on more recent versions to control for artifactual results. In particular, we investigate whether maturity of software versions can affect the original results. **Method**. We perform a re-analysis of the original data to learn and correctly deploy the tools used for data collection and analysis and to control instrumental threats that typically affect a replication. **Results/Conclusions.** Findings confirm that patterns change more frequently when they play a crucial role in the software and when in newer releases they support more advanced features.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools And Techniques—*Object-oriented design methods*

General Terms

Design, Experimentation

Keywords

Replication, Design Patterns, Theory Validation

1. INTRODUCTION

In 2012, Zhang and Budgen [13] analyzed a large amount of empirical studies on design patterns. They found scarce evidence and limited rigor in reporting empirical findings. As such, they recommended increasing the rigor of observational studies to explicit links between conclusions and reported experience on the use of design patterns.

Replications of empirical studies can help on this by increasing internal validity or external generalisation of empirical research [7]. Replicating studies by changing dataset, method, subjects, or analysis consolidates results and increases conclusions' accuracy. Replications can be performed at different levels of complexity, from re-analysis of the same data to experiment replication.

The replication we propose mines large amount of data retrieved from code repositories to investigate change proneness of design patterns. We replicated the study of Aversano et al., [2], at a later stage of maturity of the same systems. We performed what is called replication to *Control for Artifactual Results*, [8], in that we want to verify that results reflect reality and are not a product to specific software versions or experimenters. We ran an *independent external replication*, [1], [11], where the original and the new group of experimenters are completely different and independent. We use the technique of re-analysis of the original data to get confidence with the apparatus of the replication and validate the replication setup. Finally, we compare our results to find that we can only partially confirm the original results. In reporting the replication, we follow Carver's guidelines [4].

2. RELATED WORK

Whether the evolution of design patterns can provide useful insight into the maintenance process is under investigation. According to Gamma et al. [6] design patterns make software robust as they let aspects of software structure vary independently. In other words, in principle, with patterns, code changes do not propagate in the code uncontrollably. On the other hand, Posnett et al. [10] argue that simply the size of classes more than the role of design patterns determines the proneness of software classes to be changed. Given the amount of data to mine, literature often studies the evolution of design patterns of one-two consecutive versions, e.g., [2, 5]. Bieman et al. [3] is one of the few examples that investigates design patterns between software versions distant in time. They found that classes that play a role in design patterns are more change prone both for early and newer versions when the project is not only maintained by the Open Source community. For the project JRefactory completely maintained by the Open Source community, newer versions do not show a relation between the role in design patterns and their change proneness.

3. THE ORIGINAL STUDY

The major objective of the study of Aversano et al., 2007, is to empirically investigate pattern change proneness [2].

Objects. The study analyses few early releases of three Java Open Source Software projects, JHotDraw (5.2-5.4B2),

ArgoUML (0.9-0.20), and Eclipse-JDT (1.0 - 3.0) that can be classified as small, medium, large project, respectively. JHotDraw and ArgoUML are completely supported by the Open Source community.

Tools. Design patterns are collected with the tool of Tsantalis et al. [12] that runs an algorithm to match nodes in graphs by similarity scores. To determine in which snapshot a class participating to a pattern changed, the authors used a Perl script supported by a fact extractor based on the Javacc parser generator that reconstructs facts about the committed source code.

Procedure. To determine the change history of design patterns, the authors inspected the versioning system repositories (CVS) of the projects. They extracted the change sets, i.e., a sequence of file revisions that share the same author and commit note within a time window of 200 seconds, referenced as Snapshots. The authors reconstruct history of pattern evolution across snapshots.

Design. The authors introduce types of pattern change and analyse them over snapshots. Frequencies of changes across snapshots are reported by type of change and pattern. Finally, authors use box plots and statistical proportion tests across snapshots to identify patterns that mostly change and identify the type of these changes.

Findings. The paper reports findings that are statistically significant for the following research questions:

RQ1: How frequently do patterns change across releases? Patterns change more frequently when they play a crucial role for the intent of the application. The most frequently changed patterns are: Observer in JHotDraw that manages the update of a figure, Command and Decorator in ArgoUML that support the execution of new features for the user menu while Adaptors are used by interfaces of the UML meta-model to visualise them in Swing tables, and Visitors in Eclipse-JDT that are used in the navigation of Abstract Syntax Trees in Java.

RQ2: What kind of changes are different patterns subject to? Changes to the pattern interfaces make the pattern less resilient to changes. Method interface changes, or addition/deletion of methods, and addition/deletion of pattern subclasses cause higher impact.

RQ3: How much does source code co-change with patterns? Creational patterns exhibit more co-change than other patterns and patterns crucial to the application role have more co-change than other patterns. When a pattern is crucial for the application purpose, it tends to change together with a large number of other classes.

4. STUDY REPLICATION

According to Mittelstaedt and Zorn [9], this study is a Type III replication as we apply the same models, statistical methods and variables of the original work to different data. In particular, the replication data belongs to a newer version of the same software. Our replication includes a re-analysis of the original data as defined in Gomez et al. [8] to ensure that findings are independent from the procedure or setup.

4.1 Replicating design

The study replication consists of two stages: 1) re-analysis of the a subset of the original data to validate the setup and tune the tools 2) replication of the original analysis with newer versions. Table 1 illustrates the replication design according to the framework of Gomez et al. [8].

4.1.1 Re-analysis of original data

Objects. We chose to re-analyse data of JHotDraw being the smallest project. We selected the same releases of JHotDraw5 (releases 5.2 - 5.4B2) using the CVS repository. After a private communication with Nikolaos Tsantalis and the authors of the original paper, we were able to extract an identical sample.

Tools. We wanted to ensure that tools are selected and tuned as in the original work. We first used the same versions of the Design Pattern Detection (DPD) tool of Tsantalis et al. [12] and a fact extractor as in the original paper.

We then run the newest versions of the tools that provide better accuracy according to their change logs[1].

Results. Using the same versions of the tools used in the original work, we found the same number of design patterns as in the original work. This ensured that we properly configured the tools and in parallel verified the Java scripts we wrote to integrate and report the numerical findings. Using the newest versions of the tools, we found slightly different values for some patterns, though. Specifically, we have a lower number of the Prototype pattern and Observer and greater number for the Adapter-Command pattern, Decorator, and State-Strategy. Given the little difference and the better accuracy, we decided to adopt the newest versions in the replication and keep the difference in mind before drawing our conclusions.

4.1.2 Replicating the Pattern Analysis Process

To replicate the analysis process, we followed the steps in the original order.

Step 1: Detecting Design patterns. As in the original paper, we use the DPD tool version 4.5 [12].

Step 2: Reconstructing pattern evolution history across releases. We adopted exactly the same rationale for the evolution of a pattern. We assume that a pattern instance in snapshot j represents the evolution of a pattern instance in snapshot $j-1$ if and only if (i) the type of pattern is the same; and (ii) at least one of the main participant classes of the pattern is the same class in both snapshot $j-1$ and j.

Step 3: Snapshots extraction and co-change identification. We followed the same procedure to define snapshots (Section 3). As in the original study, we compute the number of snapshots where at least one class belonging to a pattern instance has changed and divided it by the total number of snapshots.

Step 4: Locating pattern changes and determining the kind of change. We used the same approach as in the original study to determine the kind of change that has been performed in a snapshot. We compare class $C_{i,j}$ at snapshot j, with the same class $C_{i,j-1}$ at the snapshot $j-1$, with the fact extractor. Once facts are identified, we build a Java script to detect the differences among facts between $C_{i,j}$ and $C_{i,j-1}$.

Step 5: Analyzing pattern co-change. A co-change is code that changes contextually with the pattern. As in the original study, we analyse co-changes in terms of lines of code (additions / deletions).

[1] `http://users.encs.concordia.ca/~nikolaos/pattern_detection.html`

Table 1: The replication design according to the framework of Gomez et al. [7]

Structure type	Structure Description	Purpose	Purpose Description
Site	Same Open Source Projects, but newer versions	Type	Control for artifactual results
Experimenters	External replication: different independent researchers	Setup	Change site and experimenters
Apparatus	Retrospective study, same analysis process, collection of design patterns, and statistical tools	Goal	To verify that results are not a product of the specific software versions used and experimenters
Operationalization	Same variables	Type of validity	Internal
Population Props	Unchanged. Design patterns are collected from the same Open Source projects		

4.1.3 Site

We considered the three OSS projects selected in the original paper. The original paper analyzed those releases in which the number of classes increased the most, actually tripled for all projects (Table 1, in [2]). Those releases pertain to the initial period of development where there are large fluctuations of number of classes. To analyse the impact of maturity on the projects, we select four consecutive recent versions in which the total number of classes either stays constant or even decreases. As such, we consider JHotDraw 7 (7.1 - 7.31, Mar2008 - Oct2009), ArgoUML (0.32 - 0.34, Jan2011 - Dec2011) and Eclipse JDT Core and UI (3.5 - 3.6, Jun2009 - Jun2010).

Counting Snapshots. As we mentioned, the different versioning systems might have caused some differences in the number of snapshots. In JHotDraw 5, for example, the CVS repository is not available anymore as it has been migrated to SVN. This migration has reduced the original number of commits. For this reason, the number of snapshots we were able to compute (124) is smaller than the corresponding number in the original study (177).

5. RESULTS

In this section we use ANOVA and proportion tests to derive our findings and compare them with the original study[2].

RQ1: How frequently do patterns change across releases? As in the original paper, the ANOVA statistical test shows that some patterns change more frequently than others: JHotDraw5 (p-value = $1.7e^{-05}$), JHotDraw7 (p-value = $6.62e^{-09}$), ArgoUML (p-value = $2.02e^{-07}$), Eclipse JDT Core (p-value = $3.1e^{-11}$), Eclipse JDT UI (p-value = $6e^{-04}$).

Table 2: Patterns that frequently change

Project	Original Paper	Replication
JHotDraw	Observer	Composite
ArgoUML	Adapter-Command	Prototype
Eclipse JDT	Visitors	State-Strategy

Table 2 shows the results of the proportion tests and the resulting most changed patterns. Again we can see that frequent changed patterns play a crucial role in the application. The role is changed with the maturity of the project. For example, in JHotDraw, Composites are the most changing patterns in newer versions. Namely, Composites are used for more advanced features as composing existing figures together. In the newer releases of ArgoUML, Prototypes are used in the advanced feature that renders (complex) figures in UML diagrams. Finally, in Eclipse JDT State-Strategy patterns implement different sorting and filtering algorithms in viewer elements without interfering with their loading and filling.

[2]Details and boxplots available at `http://goo.gl/pfOH57`

> Patterns change more frequently when they play a crucial role for the intent of the software and the role evolves with the maturity of the project to support more advanced features.

RQ2: What kind of changes are different patterns subject to? We analyse the kind of changes performed on the classes belonging to patterns. As in the original paper, we use proportion tests on the different types of changes. For all the projects, method interface and implementation changes dominate over other types of changes, but unlike the original paper in which method implementation predominates for JHotDraw and ArgoUML, proportion tests do not report significant differences among patterns of these two projects.

The tests report significant result for Eclipse JDT. Unlike the original paper, we separate the analysis between the JDT core and UI given the clear separation of concerns of the two major packages. For Eclipse JDT core, unlike the original results in which subclassing predominated, the most frequent changes are method implementation changes (p-value = $3.951e^{-05}$), method interface changes (p-value = 0.001), and additions and removals of class attributes (p-value = 0.001). Method implementations change more in the case of Observers (prop=100%, but there are only 6 instances) and Adapter-Command (prop = 80.27%), whereas method interfaces change more for Prototypes (prop = 28.9%). Additions and removals of class attributes change mainly for Singletons (prop=17.32%). Changes in method implementations for the Observer and Adapter-Command patterns are significant in the new as well as in the old releases. In newer releases, changes in method implementation also occur in patterns that support the development of the plug-in architecture. This happens for example for the Adapter-Command and Prototype patterns that allow plug-ins to be loosely coupled in the dynamic Eclipse runtime environment, or for the Factory Methods that are implemented to extend a Java interface used in plug-in new additions. In Eclipse JDT UI package, there are significant differences among patterns' changes only in method implementation (p-value = 0.01) but not in method interface changes (p-value = 0.46), or additions and removals of class attributes (p-value = 0.09). Changes in method implementation are the highest for Factory Method (prop = 87.50%).

> Eclipse JDT is the only project that exhibits clear differences among the types of changes across design patterns. Unlike earlier versions, changes in method implementation predominate the types of changes. Types of changes differ if patterns belong to Eclipse JDT core or UI package.

RQ3: How much does source code co-change with patterns?

Table 3: Patterns with larger source code co-change

Project	Original Paper	Replication
JHotDraw	Not decidable	Observer
ArgoUML	Singletons	Not decidable
Eclipse JDT	Visitors	Not decidable

We compute the number of lines that were added, removed, and modified as co-change of a design pattern. Differences are relevant for JHotDraw7 (p-value=$1.52e^{-05}$). The analysis by type of changes shows that this is due to new line additions originated with the Observer pattern. Differences in co-changes in ArgoUML are overall not significant (p-value=0.90), as well as those of Eclipse JDT Core (p-value=0.09), and Eclipse JDT UI (p-value=0.71), Table 3. The analysis by type of changes reports that for Argo UML Composite, Singleton, Factory Method patterns show more removals and modifications than other patterns, whereas Eclipse JDT, both for core or UI, has more additions / removals / modifications when Singletons add or remove their attributes. The finding on Singletons is the sole confirmation of the original results.

6. CONCLUSIONS

In this paper, we replicated an empirical study on the evolution of design patterns [2]. The aim of the replication was to determine internal validity of the original empirical results by applying the same analysis process and experiment apparatus to newer versions of software. We analysed four consecutive new versions of JHotDraw, ArgoUML, Eclipse JDT and we additionally used the old version of JHotDraw as benchmark to control for the correct instrumentation and operalisation of the work. We run the same research protocol of the original study, also with the help of the authors of the original paper and researchers that developed the original tools for data collection. Findings illustrate that patterns change more frequently when they play a crucial role in the software and the role evolves with the maturity of the project to support more advanced features.

In more mature versions, the types of changes are only clear for Eclipse JDT and the types depend on whether they refer to JDT core or UI. We have additionally found that patterns that characterise the architecture of the system (Observer and Adapter - Command used for plug-ins) change the most and change in method implementation in old as well as in newer releases. As Bieman et al. [3] hypothesised this might be related to the fact that the three projects have a different support as Eclipse JDT is maintained not only by the Open Source community

Finally, projects differ in maturity stages by the type of co-changes. Eclipse JDT mainly focuses on deleting or modifying existing lines across all design patterns. For less active projects like JHotDraw7, only specific patterns co-change with particular frequency.

Summarizing the results from the replication - in cases in which the results where statistically significant - we can report that also the maturity stage of the project has relevance on the design patterns that undergo modifications. This calls for more longitudinal studies on the nature of design patterns modifications.

Acknowledgements. We thank the authors of the original paper for the information provided and Nikolaos Tsantalis for the access to previous versions of the DPD tool.

7. REFERENCES

[1] J. P. F. Almqvist. Replication of controlled experiments in empirical software engineering - a survey. *Faculty of Science, Lund University*, 2006.

[2] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso, and M. D. Penta. An empirical study on the evolution of design patterns. In I. Crnkovic and A. Bertolino, editors, *ESEC/SIGSOFT FSE*, pages 385–394. ACM, 2007.

[3] J. Bieman, G. Straw, H. Wang, P. Munger, and R. Alexander. Design patterns and change proneness: an examination of five evolving systems. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pages 40–49, 2003.

[4] J. Carver. Towards reporting guidelines for experimental replications: A proposal. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research, RESER10, May 4, 2010, Cape Town, South Africa*, 2010.

[5] M. Di Penta, L. Cerulo, Y.-G. Gueheneuc, and G. Antoniol. An empirical study of the relationships between design pattern roles and class change proneness. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 217 –226, 28 2008-oct. 4 2008.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[7] O. S. Gómez, N. Juristo, and S. Vegas. Replications types in experimental disciplines. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 3:1–3:10, New York, NY, USA, 2010. ACM.

[8] J. N. Gómez S. G. and V. S. Replication, reproduction and re-analysis: Three ways for verifying experimental findings. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research, RESER10, May 4, 2010, Cape Town, South Africa*.

[9] a. Z. T. Mittelstaedt, R. Econometric replication: Lessons from the experimental sciences. *Quarterly Journal of Business and Economics*, 23(1), 1984.

[10] D. Posnett, C. Bird, and P. Dévanbu. An empirical study on the influence of pattern roles on change-proneness. *Empirical Softw. Engg.*, 16(3):396–423, June 2011.

[11] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo. The role of replications in empirical software engineering. *Empirical Softw. Engg.*, 13(2):211–218, Apr. 2008.

[12] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis. Design pattern detection using similarity scoring. *IEEE Trans. Softw. Eng.*, 32(11):896–909, Nov. 2006.

[13] C. Zhang and D. Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213 –1231, sept.-oct. 2012.