An Empirical Study on the Evolution of Design Patterns

Paper by Lerina Aversona, Gerardo Canfora, Luigi Cerulo, Concettina Del Grosso and Massimiliano Di Penta

Presentation by Michael Fäs for the Software Engineering Seminar FS10, ETH Zurich
About Design Patterns

- Design patterns provide standard ways to solve software design problems

- Initially, they often make the system more complex

- However, in theory, they make the system more maintainable and more resilient to changes

- Is this actually true in practice?
Empirical study to understand the evolution of design patterns in a software system

Main questions:

- How frequently are design patterns modified?
- What changes are made to them?
- How much other code is changed with them?
Overview

1. Pattern analysis process
2. Empirical study details
3. Results and discussion
4. Conclusions
5. Threats to validity
1. Pattern Analysis

What is analyzed and how is it done?
Pattern Analysis Process

Analysis is done in 4 steps:

1. Design pattern detection for every release
2. Pattern evolution reconstruction
3. Detection of pattern changes based on the CVS
   - What was changed?
   - When was it changed?
4. Analysis of pattern co-change
   - What classes not directly belonging to the pattern were changed together with a pattern?
Step 1: Pattern Detection

For the detection, an existing tool was used

- Proposed by the paper *Design Pattern Detection Using Similarity Scoring* by Tsantalis et al.

- Analyses Java bytecode

- Finds the following 10 patterns: Adapter/Command, Composite, Decorator, Factory method, Observer, Prototype, Singleton, State/Strategy, Template method and Visitor
Step 4: Co-Change

- *Pattern clients* and *pattern targets* are detected separately

- Clients: Classes depending on at least one of the main pattern classes.
  - E.g. The client of the adapter class, using functionality of the adaptee

- Targets: Classes the pattern depends on
  - E.g. The classes used by a Façade
2. Empirical Study Details

What research questions are answered?
Three open source Java systems are studied:

- **JHotDraw**
  - Framework for drawing 2D graphics
  - Actually started as a project to show the programming with design patterns
  - Classified as small system, 13.5k – 36.5k lines of code

- **ArgoUML**
  - UML modelling tool
  - Medium system, 99.5k – 159.5k lines of code

- **Eclipse-JDT**
  - Large system, 205.5k – 449.4k lines of code
Research Questions I

- **RQ1**: How frequently do patterns change across releases?
  - Do some patterns change more often than others?

- **RQ2**: What kind of changes are different patterns subject to?
  - Are some patterns more prone to a particular kind of change? E.g. Method addition/removal or method implementation changes
RQ3: *How much source code co-changes with patterns?*
   - Is there a relationship between the pattern type and the number of classes or lines of code that co-changes?

RQ4: *What is the relationship between pattern targets changes and pattern client changes?*
   - Do the patterns actually make the pattern clients more resilient to changes performed in pattern targets?
3. Results and Discussion

What was found and how can it be understood?
RQ1: How frequently do patterns change across releases?

- **Percentage over all changes:**
  - JHotDraw: 53% (94/177)
  - ArgoUML: 35% (1’923/5’525)
  - Eclipse-JDT: 25% (5’606/19’750)

- **Patterns changed most:**
  - JHotDraw: Observer, Composite
  - ArgoUML: Command/Adapter, Decorator
  - Eclipse-JDT: Visitor
RQ1: **Interpretation**

The patterns that change often, are found to play a crucial role for the particular application

- **JHotDraw: Observers**
  - Are used to update the visualization of the figures after a change

- **ArgoUML: Command/Adapter:**
  - Commands support new modelling features; Adapters are used to adapt interfaces of the UML metamodel for visualization in Swing tables

- **Eclipse-JDT: Visitors**
  - Used to navigate abstract syntax trees (AST). Concrete visitors support new code artifacts or refactoring
RQ2: What kind of changes are different patterns subject to?

- 6 kinds of changes:
  - Attributes (A)
  - Subclass addition/removal (H)
  - Method signatures or method addition/removal (M)
  - Method implementation (I)
  - More than one (U)
  - Commenting, style (O)
RQ2: Results II

- Within a system, different patterns undergo different changes (statistically relevantly)
  - In Eclipse-JDT, subclassing changes occurred more often on Factory and Visitor than on any other pattern

- But, the same pattern tends to change in different ways in different systems
RQ2: Results III

Also, the overall distribution of different change types vary greatly across the systems
  ◦ E.g. Many, many changes in method interface for JHotDraw

Interpretation:
  ◦ In the small system JHotDraw, developers did not care to avoid interface changes. New features lead to changes too.
  ◦ In larger systems, enhancement and maintenance are more present.
RQ3: How much source code co-changes with patterns?

- In some of the systems, the differences between patterns were significant
  - Not in JHotDraw
  - In Eclipse-JDT, Visitor caused more change than other patterns

- Generally, results are consistent with RQ1
  - Crucial patterns tend to cause more co-change than others
RQ4: The relationship between pattern targets and clients changes

- The following two measurements were made

\[ r_c = \frac{\# \text{ of client classes co-changed}}{\# \text{ of target classes co-changed}} \]

\[ r_l = \frac{\# \text{ of client class lines co-changed}}{\# \text{ of target class lines co-changed}} \]

- \( r_c < 1 \) indicates that a pattern effectively hides changes in targets from the clients
RQ4: Results II

Generally, all the patterns do their job
RQ4: Results III

- With some exceptions, $r_l$ is very low for all patterns and systems.
  - Significant higher $r_c$ was found only for Visitor in Eclipse-JDT.
  - This difference was not visible for $r_l$, however.

- So, when pattern targets change, the effect on the client is truly very limited.

- Patterns indeed make the system resilient to changes.
4. Conclusions

What are the take home messages?
Conclusions I

- Generally, design pattern components are changed frequently.

- Results show that change frequency and pattern co-change rather depend on the role played by the pattern in the application than on the pattern type.
Conclusions II

- Design patterns playing a crucial role for the application features are likely going to change even more often

- In this case, developers better consider pattern usage carefully
  - A wrong pattern choice would highly affect maintenance activities
5. Threats to Validity

What could be wrong with this study?
Detection precision completely dependant on the performance of the tool by Tsantalis et al.

For pattern targets and clients, a conservative set is used
  ◦ All type of associations (attributes, local variables, parameters) are considered dependencies
  ◦ Dependencies are propagated to subclasses

Only 10 (12) out of 23 design patterns proposed by Gamma et al. considered
Threats found by me

- Only Java projects
  - Results generalizable to other programming languages?
- Only 3 systems
  - Findings could be common by coincidence
  - ... or maybe they just took the 3 systems that gave the best results?
Threats found by me

- Only Java projects
  - Results generalizable to other programming languages?
- Only 3 systems
  - Findings could be common by coincidence
  - ... or maybe they just took the 3 systems that gave the best results?

- Threats to interestingness
  - Findings not very unexpected
  - Not many practical implications, except: *Use design patterns, they’re great!*
Final words

*Use design patterns, they’re great!*
Questions
Appendix

What else could be important???
The detection tool
Pattern Analysis Process

Analysis is done in 4 steps:

1. Design pattern detection for every release
2. Pattern evolution reconstruction
3. Detection of pattern changes
   ◦ What was changed?
   ◦ When was it changed?
4. Analysis of pattern co-change
   ◦ What classes not directly belonging to the pattern were changed together with a pattern?
Step 1: Pattern Detection

For the detection, an existing tool was used

- Proposed by the paper *Design Pattern Detection Using Similarity Scoring*
- Analyses Java bytecode
- Finds the following 10 patterns: Adapter/Command, Composite, Decorator, Factory method, Observer, Prototype, Singleton, State/Strategy, Template method and Visitor
Step 2: Evolution Reconstruction

- Given the patterns in two subsequent releases, which pattern corresponds to which?

- Two patterns are considered the same, iff:
  1. The type of the pattern is the same
  2. One of the two main participant classes of the pattern is the same
To see when a pattern was changed and what was changed, the CVS repository is mined.

Subsequent commits of the same author within a certain time-window are interpreted as Modification Transaction.

For a class, different change types are possible:
- Attribute changes
- Addition, removal of methods and signature changes
- Method implementation changes
- Addition, removal of subclasses
Step 4: Co-Change

- Within one *Modification Transaction*, classes not directly belonging to the pattern change too
- *Pattern clients* and *pattern targets* are detected separately
  - Clients: Classes depending on at least one of the main pattern classes.
    - E.g. The client of the adapter class, using functionality of the adaptee
  - Targets: Classes the pattern depends on
    - E.g. The classes used by a Façade