

Mining New Patterns by Learning from the Trenches

ROBERT S. HANMER, Alcatel-Lucent

MEHDI MIRAKHORLI, Rochester Institute of Technology

Pattern Mining is a scientific and research process where methods of knowledge discovery are used to find established ways of software analysis, design, implementation, and maintenance, and then describe such findings in as reusable knowledge for a given context. Over years, several traditional pattern mining techniques have been used in the community to collect tactic and specific design knowledge across different projects and organizations and present them in explicit and generic form of software patterns. This paper introduces a new dimension to the practice of pattern mining, where a set of design analysis tools and automated design discovery and knowledge mining techniques are used to mine large scale software repositories, publicly available bug reports in issue tracking software and the open information presented on the web to extract new patterns. The collective knowledge gleaned from this effort can be used to define new patterns or patterns prototype for improving software productivity. This paper presents a set of such patterns and illustrates anecdotal examples of the new patterns discovered through these techniques. Furthermore, this paper discusses the challenges faced by the pattern community in order to continue discovering, maintaining and organizing patterns in a systematic and usable way.

Categories and Subject Descriptors: **D.2.11 [Software Architectures]:** Patterns—Languages; **D.2.13 [Reusable Software]:** Reusable Library—Reusable Models

General Terms: Design, Languages, Documentation

Additional Key Words and Phrases: Pattern Mining, Design Patterns, Knowledge Discovery

ACM Reference Format:

Hanmer, R. and Mirakhorli, M. 2014. Mining New Patterns by Learning from the Trenches. ACM PLoP '2014

1. INTRODUCTION

Software patterns are an effective way to convey information during the design and construction of new systems [1]. Once the new system has been created there is still the need to understand the software in order to study or maintain the system. Patterns are useful in this context also. They are useful to exchange conceptual information and to describe the structure. The presence or absence of patterns in the software artifacts can provide insight into the overall nature of the software, the mindset of the designers and builders and the maintainability of the software.

In this paper we look at ways of mining patterns from a software artifact for analysis and maintenance purposes. In addition to the traditional techniques of interviewing humans to mine patterns, tools can be used to examine artifacts [2][3]. To do an effective job mining from the artifact however both styles of techniques for mining should be used.

This paper provides a brief review of the existing patterns for pattern mining. This is followed by several new patterns useful for general pattern mining and also for mining by automated methods.

1.1 Catalogue of Mining Techniques

There are several approaches to Pattern Mining. Traditionally the effort has focused on mining new patterns based on (1) individual contributions, (2) mining based on experts' interviews, and (3) mining through collaborative pattern mining. These techniques are primarily human centered and require several rounds of interviews or workshops to solicit the design knowledge from expert, specify the elements of a pattern and clarify the details.

1.2 Individual Efforts

Many people have collected tacit design knowledge within organizations. It is stored within an organization in the form organization-specific pattern catalogues and pattern languages.

Here are a few examples of such efforts that the pattern community will be familiar with already.

Grady Booch collection of architectural decisions [4] in his Handbook of Software Architecture is an example of such individual effort. Grady's collection includes most of the architectural decisions published across several text books, article and white papers.

Munawar Hafiz, Paul Adamczyk and Ralph Johnson [5] established a catalog of all the security patterns that have been published in the last fourteen years. This catalog has been established with the massive effort of pattern miners over 12 months period.

Bob Hanmer [6] during his last 20 years of involvement in pattern community has collected a set of architectural patterns published in his books. Moreover he has been actively specifying and organizing a collection of specific software patterns he has used over years of his architecture design experience at Bell Labs and Alcatel-Lucent.

Takashi Iba presented a set of pattern mining patterns to help pattern miners by discussing the ways to find and solve problems for pattern mining. He has also taught classes in pattern mining and engaged his whole school in pattern mining efforts as a way of enculturization.

Rick Kazman at Carnegie Mellon Software Engineering Institute have been active in identifying a new set of architectural tactics to enhance software testability, modifiability, portability[1].

1.3 Challenges:

Although several individuals' effort have resulted in great contributions to the pattern community, the focus and longevity of these resources are dependent on these individuals. The process of mining patterns has been very demanding and had required years of efforts. Furthermore there is not a single point of access or a single catalog to navigate through these patterns.

Accessibility for practitioners and educators must be created and maintained. Patterns aren't much use if they can't be found, understood and used in design or understanding. They need to be cataloged and kept up to date.

2. Pattern Mining Techniques

Webster's defines *archaeology* as the study of antiquity (Websters 2014). In the software world, where things are changing so fast, this is a fitting term for looking at software artifacts to understand how they are built and to get into the minds of their creators.

This paper contains several new patterns for archaeological pattern mining. Table 1 lists thumbnails for these patterns.

Table 1. Pattern Thumbnails

| | |
|--|--|
| 1. Software Archeology | Utilize the software analysis, design discovery and architecture reconstruction and visualization tools to mine software system's source code and detect new notions of design, hybrid approaches to a design problem and also some of the tacit forces in implementing an architectural choice. |
| 2. Crowdsourcing to extract and document design patterns | This pattern is about obtaining ideas and solutions by soliciting contribution from a larger group of experts online and in untraditional ways. This is accomplished by publishing the prototyped pattern for testing, examining it scientifically, voting on the correctness. |

| | |
|--|--|
| 3. Pattern Mining Patterns to guide the miners | Provides a roadmap for the pattern miner to extract the partial design knowledge and pull out the pattern through iterations of knowledge discovery and synthesis |
| 4. Web and Social Media Mining | This pattern used the web-content as a new resource for discovering new design patterns. |
| 5. Pattern Representation | Reintroducing a pattern, breaking the existing one into concrete, independent single forced solution or combining the single facet solutions into existing ones. |
| 6. Open Encyclopedia of Patterns (proposal) | The extracted patterns in the community can be stored in a single openly accessed pattern catalog so that experts can provide feedback, enhance, modify and use the pattern. |

2.1 Software Archeology

This section describes the Software Archaeology patterns and presents an example realization of this pattern.

Name: Software Archeology

Problem:

You have a large software artifact that you want to study to understand what patterns were used by its creators. You also want to see if there are new and interesting combinations of existing patterns that were used in its creation.

Forces:

- You could hunt down the people that created the software artifact, but in many cases you can't find out who they are, or they've moved on and aren't interested in talking about the old project or they don't have the time to really help you out. "Crowdsourcing to extract and document design patterns" could be done -- to get a group of people to jointly help
- You have access to the source code, something that you don't always have.
- The documentation about how it was implemented, the design documentation, is unavailable. It might be non-existent. Sometimes its available but you aren't quite sure if you can believe it, or it's for a previous version of the artifact.
- You aren't undergoing the effort to fix a single, concrete fault. Your goal is more of overall educational need. You want to understand the artifact to evolve it, or to assume ownership for it, or maybe to collect metrics for general software engineering research.
- You'll get different information if you examine the code "at rest" or if you examine it while it's executing. Both kinds of information are useful and complement each other.

Solution:

Utilizing design discovery techniques to extract design knowledge from source code

These techniques are

- Archie: an automated technique to detect design decisions. [Mirakhorli et. al 2012].
- Lattix, Struture 101: Structure analysis tools to discover architecture from source code
- Design Pattern Detection Tools
- Source code analysis tools

You come up with a general outline of the patterns in the artifact. But it still requires human eyes to really determine whether the pattern is useful. This is another place where "Crowdsourcing to extract and document design patterns" can come in useful.

2.1.1. Example of Utilizing Design Discovery Techniques:

In an earlier work [2], one of the authors of this paper utilized the following three steps to analyze source code of several software systems to understand how low level design decisions can be used to implement high level architectural tactics.

1. Archie [9] an automated design discovery technique to detect high level design decisions known as architectural tactic in several software systems.
2. A design pattern discovery technique [7] was used to identify the cases where architectural tactics were implemented using design patterns.
3. An overlap analysis was performed to understand forces and variability points across each tactic.

As a result of applying software archeology pattern, we were able to extract six primary reasons for adopting design patterns in scheduler implementations. Figure 1 summarizes these findings showing the patterns and their rationale for inclusion. Figure 2 depicts the structure of these new patterns.

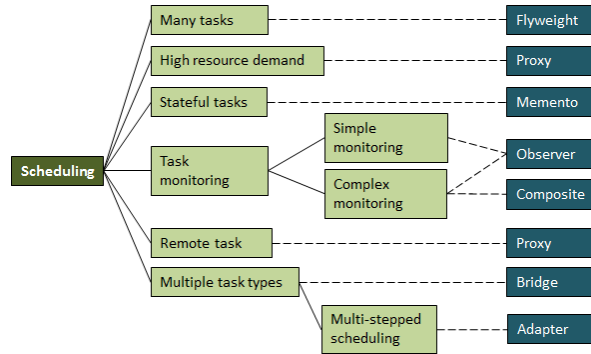


Figure 1. New Patterns for Implementing Scheduling Tactics

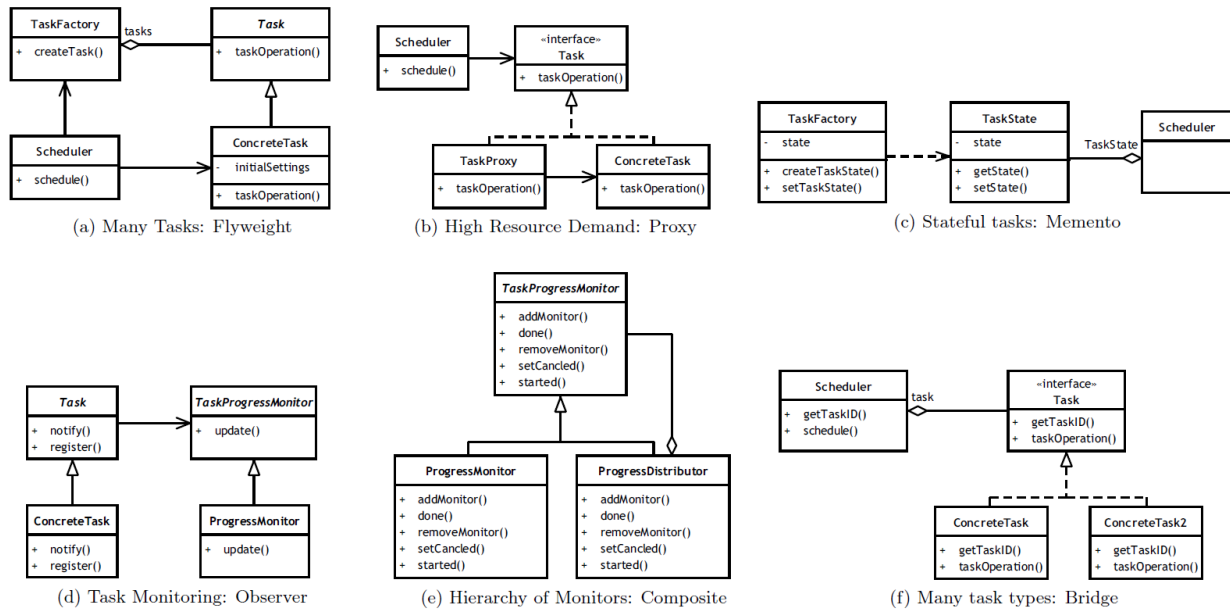


Figure 2. Design Patterns used to address variability points in the Scheduling Tactic

2.2. Crowdsourcing

The second pattern for mining new software design pattern is the Crowdsourcing Pattern described below:

Name: Crowdsourcing to extract and document design patterns

Problem: You want to find new patterns for specific software problem, but there are a number of experts you can interview and their availability are limited.

Forces:

- Traditional discovery techniques based on interviewing individual experts is difficult.
- You have limited time to go through traditional pattern mining process.

- Sometimes there are many experts, but they are all busy and you can't get on any of their calendars to interview them.
- The experts have some interest in the problem and will spend time addressing the issue, but it needs to be when they find time rather than a formal interview.
- In some cases you know who all the experts are in advance, but sometimes, for example in an open source project, you know there are experts but they are slightly anonymous.
- You have some ideas where to dig for patterns, so you are able to guide a discussion towards the topics of interest.
- Blindly asking experts can result in wild goose chases or being led down the wrong paths, so some way of deciding evaluating the ideas is needed.

Solution:

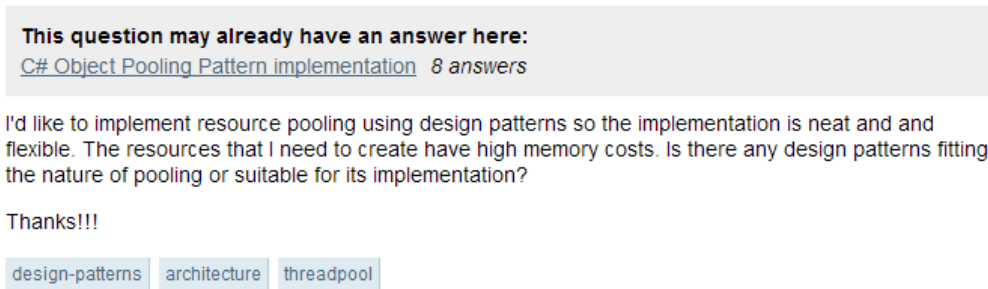
Crowdsource the problem in a relevant forum to ask the experts about how they would address the issue. Recently several of such forums and website are available where each brainstormed idea will get voted by the experts. Some examples are Stack Overflow and Stack Exchange.

2.2.1. Example of Utilizing Crowdsourcing

There are several forums and websites which the developers are actively sharing their development challenges, seek feedback from their peers and provide help to others. There are a large body of design knowledge communicated in this forums, Stackoverflow (<http://www.stackoverflow.com>) has over 16,000 posts tagged with Design-Pattern, 44,000 about different issues on software design, about 8,000 on Architecture, and so on.

Following the Crowdsourcing pattern, we posted an initial idea to Stackoverflow website¹ to mine new pattern about implementation of Pooling architectural decision. The question was worded as though asked by a developer to focus the crowd's answer to the specific question would be higher.

Figure 3. StackOverflow Post to Crowdsource the Idea



Based on the comments, feedback and suggested links of the developers on our posting, we were able to identify the following forces to consider in implementation of Pooling and also a set of solutions to address them.

Table 1. StackOverflow Outcomes

| | <u>Forces</u> | <u>Discussed Solution</u> |
|-----------------|--|--|
| <u>1</u> | Are the objects expensive to create? | FlyWeight Design pattern was suggested for creation of a large number of objects or expensive objects |
| <u>2</u> | Will they be acquired / released very frequently? | For timely pooling, Singleton design pattern was recommended. |
| <u>3</u> | Is re-initialization costly and more than a simple assignment? (eg. a pool of socket objects that must not be re-used for 5 minutes) | Pool Manager thread effectively splits the pool into a couple of puddles - those objects available for re-use and those awaiting |
| <u>4</u> | What is the resource allocation model and access strategy? Round Robin, FIFO, LIFO, Intelligent... | Different Data Structures |

¹ <http://stackoverflow.com/questions/24344877/implementing-resource-pooling-using-design-patterns>

| | | |
|----------|--|--|
| <u>5</u> | Mechanism for creating resources. | Factory Design Pattern was recommended |
| <u>6</u> | Strategies for Resource Loading (Lazy Load, Eager Loading) | NA |

2.3. Pattern Mining Pattern

Name: Pattern Mining Patterns to guide the miners

Problem: Where do you start when investigating a large body of software that's staring you in the face?

Forces:

- Software is varied, but almost all languages follow some basic rules. These rules take the form of common idioms in the language – class definitions, lexical structure, etc. These rules provide some definition to the body of code making it less formidable.
- You've done this before, or at least other people who can help you have done it before.
- You know what the software is required to do, so you have insight into some possible guiding principles that were used.

Solution:

Start with some of the well-known idioms for examining large code bases and understand the design solution. To capture a tacit design solution, the pattern miner need to observe various chains of evidences from intentions of architecture (e.g. functional requirements, quality concerns, business goals and constraints) to its actual outcomes (e.g. design decisions etc.), then synthesize these findings, reason about them, fuse them to make the tacit design solution in the code explicit.

Some suggested starting points and idioms are:

- look for the structure of the mainline code or loop,
- digging deep early is preferred to getting a broad high-level view sometimes,
- understanding the use of pre-packaged frameworks can help you relate it to similar systems.

2.4. Web mining

This pattern uses the content on the web as a resource to obtain the pattern ideas. Furthermore text mining tools and techniques can be used to assist the pattern miner in this process.

Name: Web and Social Media Mining

Problem: In the fast pace of technological changes, many organizations and experts publish their new innovative design discoveries through their blogs, websites, and webinars. A designer facing a problem wants to find if anyone has faced the same problem and has found a robust solution for it.

Forces:

- The answer to your problem is not in textbooks, your problem is domain specific.
- People like to share how smart they are with others by explaining their good ideas or successes. The source code might not be the best place to do that.
- An individual pattern contributor wants to introduce a new twist on an existing pattern that he found useful.
- A practitioner wants to explicitly publicize a new pattern which has been tacit within his/her organization.
- A developer who has recently fixed a design problem through searching the web, wants to broadcast the new pattern to his problem.

Solution: Webmining can be used to discover the new patterns. This can be done manually by using search engines or going through websites such as StackOverflow (<http://stackoverflow.com/>) to find the new patterns.

Alternatively webmining and web crawling tools as well as text mining techniques such as text classification and topic modeling can be used to perform semi-automated techniques to discover new knowledge.

2.5. Useful Patterns

Name: Pattern Representation

Problem: There are many similar patterns published, people who are new to the pattern concepts can have difficulties understanding or applying a pattern.

Forces:

- You are not developing a new pattern. An existing pattern is already written and available to you.
- Projects and organizations frequently have their own vocabulary. Pattern names should enter this vocabulary.
- Sometimes a general-purpose pattern that is widely known isn't known within a project. This can be because the terminology might not match the local vocabulary.
- A pattern is complex and can be divided into separate stand alone and simpler patterns.

Solution: An existing pattern can be synthesized to evaluate the forces and their mutual impacts. It can be rewritten into a new presentation which is simpler or more focused on a specific audience. It can include specific forces, or has a partial solution for specific forces in the original pattern tailored to the vocabulary of the project.

2.5. Pattern Wiki

Proposal: Open Encyclopedia of Patterns

Problem: There are many people who want to contribute to your understanding of a software artifact.

Forces:

- People want to contribute, but don't necessarily want to take all the time that's required to put a paper together for a pattern conference or to write a book.
- An individual pattern contributor wants to introduce a new twist on an existing pattern that he found useful.
- A practitioner wants to explicitly publicize a new pattern which has been tacit within his/her organization.
- A developer who has recently fixed a design problem through searching the web, wants to broadcast the new pattern to his problem.
- You could publish that on the internet but there is not a right place that has high visibility for pattern users. You could write a pattern paper and publish it, but it would not be easy for the pattern users search and find the pattern they want through papers.
- Design patterns are scattered across various resources, books, personal libraries, inside individual development artifacts. The lack of single point of access to the collection of these patterns make utilizing, evolving or introducing new pattern prototypes difficult.
- Pattern editing by random people can result in the main message getting confused, so some form of review is needed.

Proposed Solution: A collaboratively edited, multilingual, open encyclopedia that is supported by the non-profit group and volunteers. A pattern miner can add a new pattern to this pattern wiki, while others can provide feedback modify, enhance or add new forces and context to it.

ACKNOWLEDGEMENT

The authors would like to thank Ernst Oberortner for his contributions as our shepherd.

REFERENCES

- [1] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Addison Wesley, 2003.
- [2] Mirakhorli, M.; Shin, Y.; Cleland-Huang, J.; Cinar, M.; "A tactic-centric approach for automating traceability of quality concerns", Proceedings of the 2012 International Conference on Software Engineering, 639-649, 2012, IEEE Press.
- [3] Mehdi Mirakhorli, Patrick Mäder, and Jane Cleland-Huang. 2012. Variability points and design pattern usage in architectural tactics. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of

Software Engineering (FSE '12). ACM, New York, NY, USA, , Article 52 , 11 pages.
DOI=10.1145/2393596.2393657

- [4] Grady Booch. 2005. On creating a handbook of software architecture. In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '05). ACM, New York, NY, USA, 8-8. DOI=10.1145/1094855.1094862
- [5] Munawar Hafiz, Paul Adamczyk, and Ralph E. Johnson. 2012. Growing a pattern language (for security). In Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software (Onward! '12). ACM, New York, NY, USA, 139-158. DOI=10.1145/2384592.2384607 <http://doi.acm.org/10.1145/2384592.2384607>
- [6] Hanmer, R. S. , Patterns for Fault Tolerant Software, John Wiley, 2007
- [7] Rasool, G.; Mader, P., "Flexible design pattern detection based on feature types," Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on , vol., no., pp.243,252, 6-10 Nov. 2011
doi: 10.1109/ASE.2011.6100060
- [8] Websters. dictionary.reference.com/browse/archaeology. Referenced 15 August 2014.
- [9] Mehdi Mirakhorli, Ahmed Fakhry, Artem Grechko, Mateusz Wieloch, Jane Cleland-Huang "Archie: A Tool for Detecting, Monitoring, and Preserving Architecturally Significant Code", 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)