Reflecting Pattern Relationships in a Pattern Format

Waheedullah Sulaiman Khail
0000-0003-1494-2499 Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava

Valentino Vranić
0000-0001-9044-4593 Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava

ABSTRACT

Patterns can be applied in isolation. However, the power of patterns lies in using them in sequences. Identifying pattern relationships with other patterns is difficult. The existing pattern formats reflect the relationships with other patterns in a very informal way. We are proposing changes to the pattern format so that it reflects the relationships with other patterns in a more sophisticated way. These relationships can be used in pattern composition. We demonstrate the new pattern format on organizational patterns of agile software development.

CCS CONCEPTS

• **Software and its engineering → Patterns**.

KEYWORDS

Patterns, Pattern Relationship, Organizational Patterns

ACM Reference Format:
Waheedullah Sulaiman Khail and Valentino Vranić. 2019. Reflecting Pattern Relationships in a Pattern Format. In *24th European Conference on Pattern Languages of Programs (EuroPLoP '19), July 3–7, 2019, Irsee, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3361149.3361180

Christopher Alexander defines a pattern as [3]:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

In the last twenty years, patterns have become an integral part of software development [10, 17]. Patterns have been published in different domains of software development, e.g., as design patterns [15], architecture patterns [4, 13], analysis patterns [8] and organizational patterns of agile software development [6]. The examples used in this paper are focused on organizational patterns of agile software development as defined by Coplien and Harrison [6].

Organizational patterns can be applied to create new organizations from scratch or they can be applied to already existing organizations [6]. Organizational patterns play a crucial role in agile software development. They are not limited only to software development.

Even though patterns are known from late 70s [3], there are still difficulties in selecting the right pattern for the problem in hand [9]. To fully utilize and employ an appropriate organizational pattern, one must thoroughly understand not only the pattern itself, but also the pattern language it belongs to [2**?**]. Furthermore, pattern languages in general do not always reveal every plausible connection with other patterns. Understanding organizational patterns and their connections is a difficult task by itself. Application of appropriate patterns might therefore be challenging.

Patterns are organized in pattern languages. Pattern languages provide means to connect patterns together, which assist in solving bigger problems that occur together. However, pattern languages do not reveal all the information on how the patterns they consist of are connected to each other.

Even though it is not uncommon to employ a single pattern, the power of patterns lies in applying them in combination with each other. The nature of patterns is such that multiple patterns will be applied together. A pattern application generates a resulting context which matches the context of some other patterns. This way more patterns are composed together and progress is made towards a whole or bigger solution. But patterns can be composed at varying levels.

The order in which to apply patterns is critical to a useful pattern composition. Patterns connected with each other form a network, where patterns are typically applied in combination with each other [2–4, 17]. These connection means are used to select the right patterns for the problem in hand and navigate to the next pattern to create a sequence of patterns. Sequences represents the orders in which the patterns should be applied in order to solve a specific problem in hand [5, 12]. Sequences are gradually emerging from successive application of patterns in an appropriate order. Pattern sequences demonstrate the proper use of a pattern language. Various sequences of a pattern language show different solution to many problems that these patterns can resolve.

The relations between patterns in a pattern language are defined on the semantic level. In order to create useful sequences of patterns, we need to identify and understand these relationships. There are no specific sequences or rules identified that would help with deciding which pattern to apply first and which one next [**?**]. Doubts in choosing specific pattern sequences or incorrect decisions might slow down productivity rate of the organization and slow down the development process.

Pattern authors put a lot of effort in writing their patterns. However, if readers do not understand patterns or the links between

EuroPLoP '19, July 3–7, 2019, Irsee, Germany

ACM ISBN 978-1-4503-6206-1/19/07…$15.00
https://doi.org/10.1145/3361149.3361180

them, then it will be difficult to apply them in sequences. Numerous patterns have been documented. However, they are not well connected [11].

It is important to note that there are direct and indirect relationships between patterns in a pattern language.

However, the existing pattern formats do not clearly reflect this. Consequently, the relationships between patterns are very difficult to recognize. This difficulty is creating uncertainty during pattern application.

In order to recognize the relationship between patterns, we propose changes the pattern structure that will reflect the relationship links. We propose adding subsections called "keywords" to the problem and solution sections of a pattern. The keywords subsection in the problem section of a pattern includes relationship links (keywords) to all the patterns that can be applied before this pattern. While the keywords subsection in the solution section of a pattern includes links to all the patterns that can be applied after this pattern. These relationship links can be used for creating proper pattern sequences.

The rest of the paper is structured as follows. In Section 1, we look into the structure of a pattern in different pattern formats. In Section 2, we identify relationship links in a pattern language. In Section 3, we propose our solution. In Section 4 we draw conclusion and outline our further work.

1 PATTERN FORMS AND PATTERN STRUCTURE

Many pattern authors tend to write patterns in their own format, but there are some common pattern formats (often denoted as *pattern forms*). Although the pattern formats differ, the core structure is the same. Most of the patterns formats embrace these essential elements:

- **Pattern name**. The name of the pattern is a single word or short phrase to refer to the pattern.
- **Context**. The context represents the current situation under which the problem and its solution seem to happen. It is a precondition for the pattern's applicability. It represents the initial configuration of the system before the pattern is applied to it.
- **Problem statement**. The problem statement explains what is the problem and why the problem needs a sophisticated solution. The problem statement characterizes the essence of the pattern. The problem statement also includes description of the conflicting forces or may be stated solely through conflicting forces, which is the way Alexander did it, and which was adopted by Coplien and Harrison [6].
- **Solution**. The solution part describes the design which will put the conflicting forces identified in the problem statement into a balance.
- **Discussion/Consequences**. The discussion/consequences section deals with the trade-offs of applying the pattern.

Alexander's original format (or Alexandrian form) [2, 3] includes also a picture of the pattern. The GoF format [15] splits the problem and solution sections into more detailed subsections. Additionally, the GoF format includes an explicit section for known uses of a pattern, example code, and related patterns. Coplien's format (or Coplien form[1]) includes the related patterns section. In the Portland format [16], the problem statement is described in a couple of paragraphs, followed by the word "Therefore," which is followed by a couple of paragraphs where the solution is discussed. Patterns written in the Portland format are usually short. The POSA format![4] is quite similar to the GoF form with different headings. Patterns written in the POSA format tend to be large, spanning over several pages. They include a summary section at the end.

Table 1 compares the pattern formats. The current structure of a pattern is very well narrated. However, the relationship with other patterns are not very well reflected. As mentioned above some pattern formats include a specific related pattern sections. In the related patterns section, those patterns are specifically mentioned, which can be used together in sequences with this pattern. This helps in identifying the relationship between patterns. However, the pattern relationships to other patterns should not be limited to those patterns in related patterns section.

2 IDENTIFYING RELATIONSHIP LINKS IN PATTERNS

In order to apply patterns in sequences, we need to identify relationship between patterns. Existing pattern formats reflects the direct relationship between patterns. Direct relationships are those references to other patterns introduced namely in the pattern description. In particular, the *Solution* and *Related Patterns* sections are intended to discuss those patterns that can be used in sequences with the pattern being described. Although these direct references are helpful in identifying pattern relationships with other patterns, they are limited to the patterns known to the pattern author. Another limitation is the relation with those potentially related patterns, which will be observed later.

There are other, indirect relationships between patterns. These indirect relationships can be found in the pattern format. Indirect relationships are identified by describing areas of interest of patterns in a pattern language. An indirect relationship can help in identifying related patterns. However, these indirect relationships are hidden within the pattern description. It is hard to identify them in a pattern. In the following example pattern *Size The Schedule* [6] we first identify indirect relationships, and later we will categorise these indirect relationships into two types.

We look into the problem section:

> Both overly ambitious schedules and overly generous schedules have their pains, either for the developers or the customers.
>
> If you make the schedule too generous, developers become complacent, and you miss market windows. But if the schedule is too ambitious, developers become burned out, and you miss market windows. And if the schedule is too ambitious, product quality suffers, and compromised architectural principles establish a poor foundation for future maintenance.
>
> … it's clear that schedule and functionality trade off against each other.

[1] http://wiki.c2.com/?CoplienForm

Pattern Form	Pattern Name	Diagram	Context	Problem Statement		Solution	Discussion	Consequence	Related Patterns	Known Usage
				Problem	Forces					
Alexandrian	√	√	√	√	√	√	√	X	√	X
GOF	√	X	√	√	√	√	√	√	√	√
Coplien	√	√	√	√	√	√	√	X	√	X
Portland	√	X	X	√	√	√	X	X	√	√
POSA	√	X	√	√	√	√	√	√	√	√

Table 1: Pattern formats.

> The customer believes you can cut functionality, but a promise of having the yet unattained functionality at some future date leaves the customer much less comfortable. And projects without schedule motivation tend to go on forever, or spend too much time polishing details that are either irrelevant or don't serve customer needs.

There are direct references to other patterns from different pattern languages in this excerpt. At the moment, we ignore them in order to focus on identifying indirect relationships. For this, we identify those keywords which can potentially be used as relationship links to other patterns. These keywords (relationship links) can be called indirect relationship. *Size The Schedule* is an organizational pattern of agile software development, hence we search for keywords from agile glossaries [1, 14].

In patterns from a different domain, the selection of keywords can be different Based on the keyword from agile glossaries we identify: schedule, market window, product quality, architectural principles, maintenance, functionality, trade of, and project as indirect relationship keywords.

The same can be done with the solution part of the pattern. There are direct references to other patterns. The solution part also indicates terms and keywords, which can be used as indirect relationship links between this pattern and other patterns. Here's the corresponding excerpt [6]:

> Reward developers for negotiating a schedule they prove to meet, with financial bonuses; see *Compensate Success*, or with extra time off. Keep two sets of schedules: one for the market, and one for the developers.
>
> The external schedule is negotiated with the customer; the internal schedule, with development staff. The internal schedule should be shorter than the external schedule by two or three weeks for a moderate project. If the two schedules can't be reconciled, customer needs or the organization's resources–or the schedule itself–must be re-negotiated *Recommitment Meeting*.
>
> Help delineate the schedule with *Named Stable Bases*. Grow as needed with *Phasing It In*. Define initial targets with *Work Queue*. Make sure *Someone Always Makes Progress*
>
> You don't need a full schedule–perhaps no schedule at all–to get started. See *Get On With It* and *Build Prototypes*.

The keywords that can be observed in this excerpt are: reward developer, negotiate schedule, extra time off, external schedule, internal schedule, schedule reconciliation, customer needs, initial target, and grow as needed.

Direct references to other patterns are not complete. This means not all linked or related patterns are referred to directly. In other words, the directly referred patterns are not the only related patterns that can be used in sequences with a pattern. Otherwise this will limit usage of the pattern in sequences with other patterns. These keywords extracted from the problem statement and solution part can be termed as indirect relationship links between patterns. They help in creating sequences in a pattern language and across pattern languages.

By analyzing Coplien and Harrison's organizational patterns [6], we categorize the identified keywords. If we look into the forces in the problem statement they discusses the burning issues, and who caused these issues. We look again into the *Size The Schedule* pattern:

> If you make the schedule too generous, developers become complacent, and you miss market windows. But if the schedule is too ambitious, developers become burned out, and you miss market windows.

There are two conflicting forces: too generous schedule versus too ambitious schedule. The forces indicate that there is a dependency between this pattern and other related patterns[4]. We will come back to this dependency shortly.

According to the context of the *Size The Schedule* pattern, it is most often used at the start of a project. However, the dependencies in the problem statement indicates that the *Size The Schedule* pattern can be applied at any stage of the project. There are three closely related issues being debated in the forces section of the pattern *Size The Schedule*. First a "missed market window" or perhaps a delay in the project. Second the developer is involved in the forces by either "developers burned out" or "developers become complacent." Third the subject in this case the (schedule) is either "too ambitious" or "too generous".

Now we look into dependencies.

The problem section explains what is the problem and why the problem needs a sophisticated solution. Hence we look into the reasons that why this problem has happened, by analyzing the problem section from *Size The Schedule* pattern.

The first two issues debated in the forces section, "missed market window", and "developers become complacent" vs "developers burned out" indicates that these problems can be caused by either changes in the developers schedule, or changes in developers priorities or tasks. This means some other patterns have potentially been applied which was related to developers schedule, or developers tasks. At the same time the two variants of a schedule "too ambitious" vs "too generous", indicates that any other pattern related to project schedule can caused this problem.

The above mentioned dependency and other such dependencies are revealing that the *Size The Schedule* pattern might not always be the first pattern in the sequence. There can be many other patterns before this one. This implies that the indirect relationships mentioned in the problem statement are links to those related patterns that can be applied before this pattern in a pattern sequence.

Now we look into the solution section of the *Size The Schedule* pattern:

> Reward developers for negotiating a schedule they prove to meet, with financial bonuses; see *Compensate Success*, or with extra time off.

The solution section always speaks about things to be performed. This indicates that indirect relationships mentioned in the solution section, identifies those related patterns which can be used after this pattern in a pattern sequence. For example rewarding successful developers with financial bonus or extra time off. The terms reward developers, financial bonus, extra time off are all pointing to other related patterns. The solution section often mention directly related patterns. However in order not to limit the usage of this pattern, these indirect relationships can be used as relationship links with other patterns.

3 REFLECTING RELATIONSHIP LINKS IN PATTERNS

Identifying these relational links is difficult because one have to understand the pattern completely. Only by then we will understand why a specific terms was used. However, the pattern authors know exactly why a specific term is used. At the same time, the author can identify which terms are linking a pattern to other patterns. We propose that the authors add a subsection entitled *Keywords* both to the problem statement and solution section. In these subsections, all those relationship links can be explicitly mentioned.

When applying a pattern, developers or pattern users will easily find other related patterns through these relationship links. These relationship links help pattern users to create pattern sequences across pattern languages in the relevant domain.

For example, the keywords subsection in the problem statement of the *Size The Schedule* pattern would look like this:

> Keywords: developer, customer, schedule, market window, product quality, architectural principles, maintenance, functionality, trade off.

The keywords subsection in the solution section would be:

> Keywords: reward developer, negotiate schedule, extra time off, schedule reconciliation, customer needs, initial target, grow as needed, make progress.

These are the possible relationship links between this pattern and other patterns. We extract relationship links for three other organizational patterns from Coplien and Harrison's and Cunningham's catalogs [6, 7].

The keywords subsection in the problem statement of the *Compensate Success* [6] would look like this:

> Keywords: schedule, organization, ego-less team, make or break projects, reward team, investment in speculative work

The keywords subsection in the solution section would be:

> Keywords: comparable rewards, exceptional awards, celebration, customer satisfaction, systemic success, food culture, discourage peers, individual contributions, successful projects.

The keywords subsection in problem statement of *Recommitment Meeting* [6] would look like this:

> Keywords: schedule, work queue, scheduling problems, high productivity.

The keywords subsection in the solution section would be:

> Keywords: interested management, development team, plan, commitment to schedule, allocating business resources.

The keywords subsection in the problem statement of the *Comparable Work* pattern [7] would look like this

> Keywords: commit to delivery, schedule, work portion, deadline.

The keywords subsection in the solution section would be:

> Keywords: work queue, later release, re-commitment, head-roam.

The diagram in Figure 1 maps the keywords that we have identified as relationship links. Based on the identified keywords, we can create sequences of related pattern. There is no limitation on which pattern can be applied first. Once we apply a pattern, the keywords in the solution section should be mapped to the keywords in the problem section of the related patterns. The keyword subsection in the solution section will definitely have more than one option. This means more than one patterns can be applied after this pattern. We can select the one based on the problem in hand.

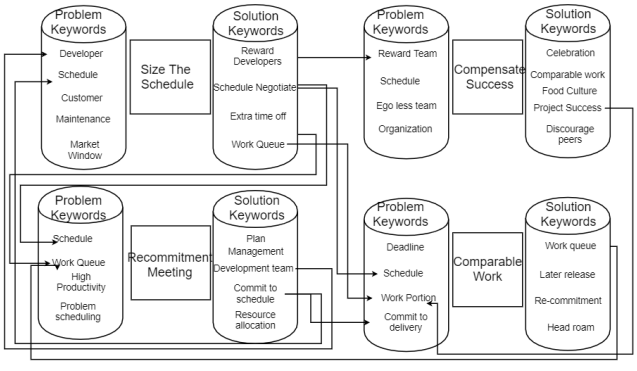

Figure 1: Indirect relationship links in four organizational patterns.

4 CONCLUSIONS AND FURTHER WORK

To create sequences of patterns, it is important to know how patterns are related with each other. In order to identify pattern relationships we propose to modify the pattern structure. Pattern authors can include the proposed *Keywords* subsection in the problem statement and solution section.

The proposed modification in the pattern format will make patterns more readable. The keywords declared in the *Keywords* subsection can act as relationship links between patterns. With these terms, we can identify which patterns can be used together in sequences.

In the future, we would like to formalize the relationship between patterns. Once pattern relationships have been formalized, we can create pattern sequences across pattern languages.

ACKNOWLEDGMENTS

We would like to thank Ruslan Batdalov for being our shepherd. Our sincere thanks also go to our writer's workshop group members namely, Ömer Ulludağ, James Lear, Frieder Jacobi, Haruka Iba, Karin Iwata, Hinako Ando, Rei Kono, Aleksandra Vranić and Kohki Ogawa, The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under grant No. VG 1/0759/19. This contribution/publication is also a partial result of the Research and Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

REFERENCES

[1] Agile Alliance. 2019. Agile Glossary. https://www.agilealliance.org/agile101/agile-glossary.
[2] Christopher Alexander. 1979. *The Timeless Way of Building.* Oxford University Press.
[3] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Joaquim Romaguera i Ramió, Max Jacobson, and Ingrid Fiksdahl-King. 1977. *A Pattern Language.* Gustavo Gili.
[4] Frank Buschmann, Kelvin Henney, and Douglas Schimdt. 2007. *Pattern-Oriented Software Architecture: On Patterns and Pattern Language.* Vol. 5. Wiley.
[5] James O Coplien and Gertrud Bjørnvig. 2011. *Lean Architecture: for Agile Software Development.* Wiley.
[6] James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development.* Prentice-Hall.
[7] Ward Cunningham. 1996. Pattern Languages of Program Design 2. Addison-Wesley, Chapter EPISODES: A Pattern Language of Competitive Development, 371–388.
[8] Martin Fowler. 1997. *Analysis Patterns: Reusable Objects Models.* Addison-Wesley.
[9] Tomáš Frťala and Valentino Vranić. 2015. Animating Organizational Patterns. In *Proceedings of 8th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, ICSE 2015 Workshop.* IEEE, Florence, Italy.
[10] Gregor Hohpe, Rebecca Wirfs-Brock, Joseph W. Yoder, and Olaf Zimmermann. 2013. Twenty Years of Patterns' Impact. *IEEE Software* 30, 6 (Nov. 2013), 88, 84–87.
[11] Gunter Mussbacher, Michael Weiss, and Daniel Amyot. 2007. Formalizing Architectural Patterns with the Goal-Oriented Requirement. In *Proceedings of 5th Nordic Conference on Pattern Languages of Programs, VikingPLoP 2006.*
[12] Ronald Porter, James O. Coplien, and Tiffany Winn. 2005. Sequences as a Basis for Pattern Language Composition. *Science of Computer Programming* 56, 1 (2005), 231–249.
[13] Mark Richards. 2015. *Software Architecture Patterns.* O'Reilly.
[14] SolutionsIQ. 2019. Agile Glossary. https://www.solutionsiq.com/agile-glossary.
[15] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.
[16] Hironori Washizaki, Suthinan Thanintranon, Masashi Kadoya, Yoshiaki Fukazawa, Takeshi Kawamura, and Joseph W. Yoder. 2014. Analyzing Software Patterns Network Obtained from Portland Pattern Repository. In *Proceedings of the 21st Conference on Pattern Languages of Programs (PLoP '14).* The Hillside Group, USA, Article 8, 6 pages. http://dl.acm.org/citation.cfm?id=2893559.2893567
[17] Uwe Zdun. 2007. Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. *Software: Practice and Experience* 37, 9 (2007), 983–1016.