Treating Pattern Sublanguages as Patterns with an Application to Organizational Patterns

WAHEEDULLAH SULAIMAN KHAIL and VALENTINO VRANIĆ, Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava

Organizing people is very important and one of the great challenges, and in particular in software development. Organizational patterns are the key to piecemeal growth of organizations. To deal with the complexity of choosing right pattern sequences and understanding pattern languages in general, we propose representing them as patterns. Such summary level patterns can be used to treat meaningful parts of pattern languages: pattern sublanguages. We applied this approach to organizational patterns. Specifically, we expressed the pattern story of establishing a new project as the *New Project* pattern. We also captured the dynamics of this pattern by a state diagram. As it can be observed by comparison, summary level patterns overcome patterns stories in terms of comprehensibility and consistency, with the main contribution being a direct treatment of the conflicting forces.

CCS Concepts: •**Software and its engineering** → **Patterns;**

Additional Key Words and Phrases: patterns, organizational patterns, pattern language, pattern sublanguage

ACM Reference Format:

Waheedullah Sulaiman Khail and Valentino Vranić. 2017. Treating Pattern Sublanguages as Patterns with an Application to Organizational Patterns. *EuroPLoP* (July 2017), 12 pages.
DOI: https://doi.org/10.1145/3147704.3147710

1. INTRODUCTION

Organizing people is very important and one of the great challenges, and in particular in software development. Organizational patterns are the key to piecemeal growth of organizations. They can be applied to correct a specific problem within an organization or to build a new organization from scratch [Coplien and Harrison 2004]. Organizational patterns are also seen as the basis for agile software development [Washizaki et al. 2014].

Although isolated application of one or several patterns is not uncommon in practice, building an organization with patterns requires understanding and employing a whole pattern language, with possibly dozens of complexly related patterns. Organizational patterns are no exception to this. At the same time, even understanding individual organizational patterns as such is difficult [Frt'ala and Vranić 2015]. If organizational pattern languages are to be accepted and applied by a wider community, a compressed form of expressing whole organizational pattern languages is necessary.

In this paper, we demonstrate how the pattern format can be applied once more to capture whole pattern languages and make them more comprehensible for purposes of easier finding of appropriate pattern sequences for the problem at hand. Moreover, we dare to claim that this is not a mere application of the pattern format and that pattern languages can actually be perceived as patterns, as others have observed, too [Buschmann et al. 2007]. More precisely, we propose not to target whole pattern languages, but their meaningful parts that can be seen as sublanguages.

The rest of the paper is structured as follows. Section 2 explains the notion of a pattern sublanguage. Section 3 brings in a pattern for the new project organizational pattern sublanguage. Section 4 discusses the findings. Section 5 emphasizes connections of our findings to related work. Section 6 draws some conclusions and outlines further work.

2. PATTERN SUBLANGUAGES

Simply stated, a pattern language comprises a set of patterns and rules of how these patterns can be applied in a sequence to achieve a certain goal [Coplien and Harrison 2004]. This definition comprises the lexical level, at which a pattern language is seen as a set of patterns, similarly as a natural language is seen as a set of words. It also comprises the syntax level or, to be more precise, the generative syntax level: the rules of how the patterns can be applied, which corresponds to the rules of how the words can be put into valid sentences in a natural language, i.e., a (generative) grammar. There is also the observable syntax level: all valid sentences themselves—i.e., all *pattern sequences* in a pattern language—also define the syntax of a pattern language. Figure 1 shows one pattern sequence for establishing a new project organizational structure based on the organizational patterns identified by Coplien and Harrison [Coplien and Harrison 2004].

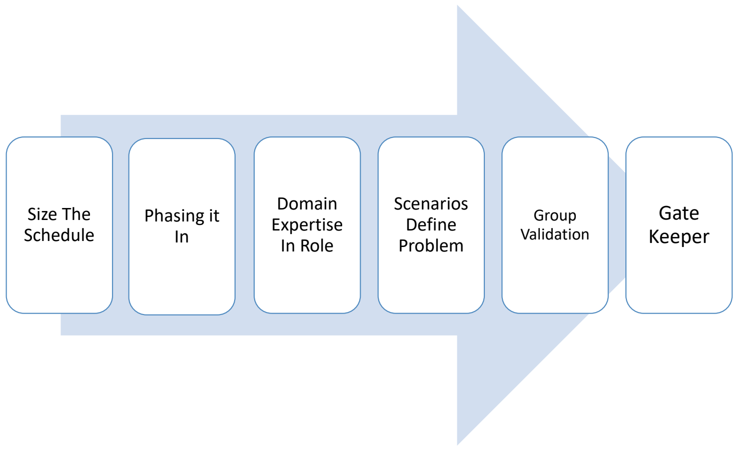

Fig. 1. A new project pattern sequence.

As with natural languages, people operate rather at the observable syntax level learning it on the examples of valid sentences and adopting the language grammar rules implicitly. Pattern languages are no exception to this, so a common practice to demonstrate a proper use of a pattern language is by pattern sequences, which represent the order in which the patterns should be applied [Porter et al. 2005].

If the number of the words in a sentence is not limited, a natural language contains an infinite number of sentences. The same holds for pattern languages, but since these are smaller than natural languages, it is easier to identify characteristic pattern sequences. Pattern sequences are inevitable part of a pattern language description. However, pattern sequences do not provide all the details of the reasoning behind choosing a particular order of patterns [Porter et al. 2005].

Pattern sequences are usually backed by *pattern stories*, which are based on specific examples originating in the experience of the story tellers [Buschmann et al. 2007; Henney et al. 2005]. An example of this is Coplien and Harrison's pattern story about piecemeal growth [Coplien and Harrison 2004, p. 132], which describes what organizational patterns were applied to build an appropriate organizational structure during a particular project realized in 1980s. Here is an excerpt:

> When I started to plan the Q project, I wanted small core team of architects, so I employed *Size The Organization* with an eye on *Phasing It In*. The project was too large for a *Solo Virtuoso* approach—though we would use that pattern later to flesh out a prototype. I put forward the opportunity and made it possible for people to sign up.
>
> My main job as project coordinator was to put up the *Firewalls* to management until we had our act together. We brought in Lalita for her work in scripting languages and their environments; Peter for his architectural expertise. Later we decided we needed market domain knowledge, and that's when we brought on Jim and Beki in the interest of having *Domain Expertise In Roles*. The recruitment strategy was always one of ferreting out matches of interest that would excite the players, amplified by the new nature and somewhat subversive approach of the opportunity. Team pride was an emergent property of this process. We also had our own value system and model of rewards: all team members would share credit for any patents that were issued, and we would seize a leadership role in the organization. We also knew we were catering to the organization's product interests, and that would be rewarded: *Compensate Success*.
>
> Beki served as the *Gate Keeper*, bringing in ideas from the AOL Instant Messenger world, interviewing (child!) users of the system, and bringing in knowledge of the organization and market opportunities. She and I split duties of *Matron Role*.
>
> We moved forward on design using CRC cards to formulate an architecture, employing *Scenarios Define Problem* and *Group Validation*. The goal was to get the project "running" on CRC cards and then to implement a first, simple cut in a one- or two-day programming session, all together in one room, doing *Developing In Pairs*.

As can be seen, the pattern story about piecemeal growth does not explicitly discuss the patterns involved, nor does it provide the reasons for their use. Moreover, no strict order of the patterns is specified by the story. Thus, this pattern story—and perhaps many other pattern stories— cover more than just one pattern sequence: a smaller pattern language embedded in a bigger pattern language. We will denote such a pattern language as *pattern sublanguage*. This notion may be compared to jargons in natural languages. Indeed, the pattern story about piecemeal growth is intended to provide some insight into the piecemeal growth organizational pattern language, one of the four organizational pattern languages defined by Coplien and Harrison [Coplien and Harrison 2004]. These may be considered being a part of the overall organizational pattern language, i.e., its sublanguages. Furthermore, many claimed-to-be pattern sequences may actually be pattern sublanguages.

A pattern language has the structure of a network [Alexander et al. 1977] formed by the links between the patterns in this language. These links determine the possible order of pattern application and are usually backed by some reasoning. These networks are usually visualized as graphs, but they usually embrace only the most important links. Thus, pattern sublanguages could be seen as subgraphs of the pattern languages they are part of if the pattern language graphs would embrace all the links between the patterns. Otherwise, new links are usually identified based on the information provided in the patterns themselves or other accompanying descriptions. Figure 2 shows the new project organizational pattern sublanguage, which is a part of the piecemeal growth organizational pattern language published by Coplien and Harrison [Coplien and Harrison 2004]. The links in this diagram are based on the organizational patterns and their relationships identified by Coplien and Harrison going beyond the links they depicted in the project management organizational pattern language graph.

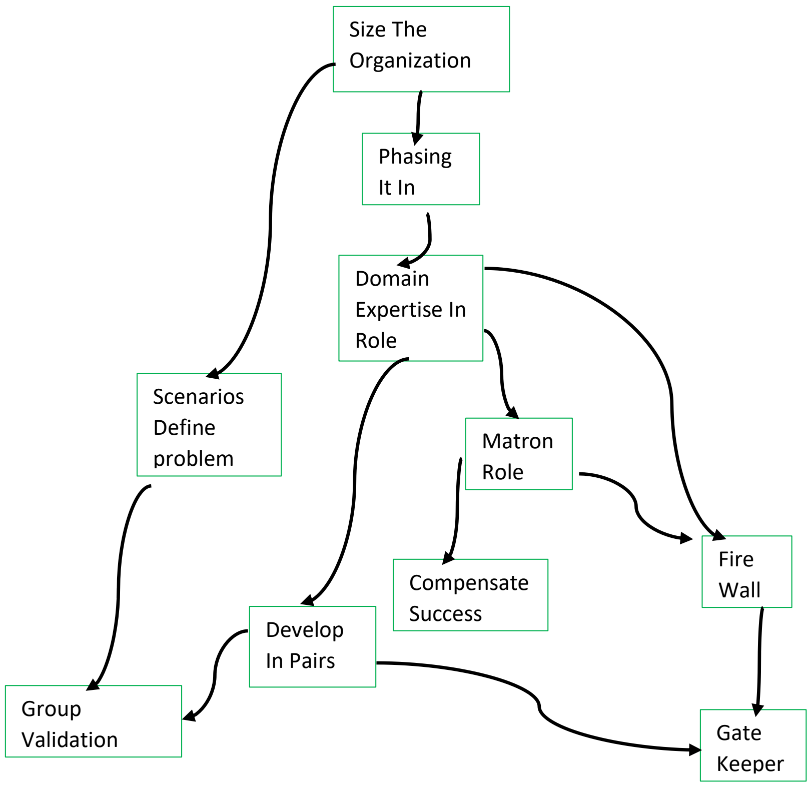

Fig. 2. The new project organizational pattern sublanguage.

Applying a pattern language or sublanguage requires one to understand the patterns it embraces. This is not easy because of the number of patterns each of which is provided with a description of several pages. For example, in the piecemeal growth pattern language [Coplien and Harrison 2004], there are 32 patterns and some of these patterns even repeatedly occur in one pattern story and the corresponding pattern sequence.

A nontrivial problem that requires a pattern-like solution can be seen as a conflict of forces [Alexander et al. 1977; Coplien and Harrison 2004; Alexander 1979]. A pattern then resolves the conflicting forces by specifying the conditions that should hold for the proposed solution to be viable [Schumacher et al. 2013; Buschmann et al. 2007]. How to apply the patterns in a pattern sublanguage undoubtedly is a nontrivial problem that might be seen as involving all the forces from all its patterns [Buschmann et al. 2007] bringing us to the idea of treating pattern sublanguages as yet another kind of a pattern: *summary level patterns*.

3. THE NEW PROJECT SUMMARY LEVEL PATTERN

The excerpt from the pattern story about piecemeal growth introduced in the previous section is not just an excerpt. Although extracted at the lexical level, i.e., by just picking out the relevant sentences from the overall story about piecemeal growth, this is a pattern story within a pattern story that addresses a well-defined motif of establishing a new project. In this section, we express this smaller pattern story about establishing a new project as a summary level organizational pattern.

Following the convention of Coplien and Harrison's organizational pattern catalog [Coplien and Harrison 2004], the pattern is written in the Alexandrian form, which is the original pattern form established by Christopher Alexander in his seminal book The Timeless Way of Building [Alexander 1979]. As can be devised from Coplien and Harrison's description [Coplien and Harrison 2004, p. 14], the Alexandrian form looks like this:

[Pattern Name]

[Context]

[Problem Statement]

Therefore:

[Solution]

❖❖❖

[Discussion of the Solution]

The names of the parts do not appear explicitly in pattern descriptions. The discussion of the solution may include links to other patterns.

The pattern itself is presented in Section 3.1. Inevitably, the pattern repeats or restates the ideas of the underlying patterns as they have been expressed by Coplien and Harrison [Coplien and Harrison 2004]. We also capture the pattern dynamics using a state machine diagram presented in Section 3.2.

3.1 The Pattern Itself

New Project

. . . the company has got a new project. Now the project needs to be kicked off. New professionals capable of making a good product have to be recruited.

❖❖❖

The aim is to build a team and an environment which will result in a good quality product.

Software development exhibits many complexities (logical complexity, project size, interface complexity, etc.) [Blackburn et al. 2006; Heričko et al. 2008]. Many researchers recommend using a small and a sharp team. However, in most cases, looking at the software project size and the need to submit the project on time, often is not feasible to have a small team [Blackburn et al. 2006]. It has been proven that there must be an ideal team size based on the project size [Blackburn et al. 2006]. Software complexity requires an increased team size, but greater team size significantly decreases its productivity.

Bringing more staff right at the start will result in a bigger team. Having a bigger team means breaking the project into modules and assigning sub teams to work on these modules. This requires more communication links, which, in turn, decreases productivity. However adding people late to the software team makes the software delivery even later [Brooks Jr. 1995]. Bringing new people late to the project team will not help in finishing it earlier or on time; this will even engage the current staff in training them or making them used to the environment and the project itself.

There will be empty roles. Either someone leaving the project midway or the task is demanding an additional role. Thus staff are not interchangeable. One cannot be a professional in all fields. There have to be a professional for each role. We cannot expect one developer to do all the tasks.

Once the team is built, its members should be kept motivated to work effectively on their tasks. Looking after the team and giving them the feeling that they belong gives them extra energy and motivation.

The communication between the developers and customer is crucial for a project to be successful because design documents alone often do not fully reflect the customer needs. However, distraction and too many noise need to be addressed professionally through a good interface.

The ultimate goal in a project is to get a quality final product. Most of the time, efforts are put in, but the final product does not comply to expectations. Sometime, companies rely on the updates from the market and bring new changes accordingly, but this is not always in the scope of what the customer expects. Regular product quality assessment is needed.

All in all, there are several problems in establishing a new project, each of which is driven by a set of conflicting forces:

—There must be an ideal team size based on the project size:
  —Small and sharp teams provide better productivity
  —Software size and complexity demands for an increased team size
—Fill empty roles as early as possible:
  —Adding people late to the project makes project even later
  —All roles must be filled with professionals
  —Not just anyone should be hired
—Staff should be given the feeling that they belong:
  —The budget has its constraints
  —Extra motivated staff contribute more than they are scheduled
  —Extra contributions inspire the whole team
—Keep good balance in the communication and interaction between stakeholders and developers:
  —Requirement change is a habitual activity in software development
  —Design documents are not very communicative
  —Many interactions with stakeholders cause distraction to developers

Therefore:

Start with hiring the team, but do not make it big right from the start. Also, do not shrink the team very much despite the complexity of the project. Thus, a medium size team should be created, so first apply the *Size The Organization* pattern.

Once the team is built, if you still need some roles to be filled, you can apply the *Phasing It In* pattern so you can gradually increase the team size ensuring that the team works well before adding new members. However, do not hesitate too much: it is better to bring new staff earlier rather than too late.

You cannot embrace just anyone to fill the roles simply because you need more staff. Thus, when we you apply the *Phasing It In* pattern, you should also apply the *Domain Expertise In Roles* pattern, ensuring that newly hired staff are professionals and domain experts that correspond to the roles you have to fill in.

The project team needs to update market and customer information and contact, but too many contacts and too much information flowing directly to developers acts as distraction and noise. Thus, you need to keep a balance, preferably by using the *Gate Keeper* and *Firewalls* patterns. By *Gate Keeper* you establish an interface to bring the market updates and information flow to the team in a professional manner and to communicate with the stakeholders as well. The *Firewalls* pattern then limits a distraction to the developer, which will prevent interrupts and unwanted noise. The *Gate Keeper* pattern facilitates effective flow of information, while the *Firewalls* pattern restricts the flow of detracting information and keeps the balance between stakeholder and developer interaction.

The team should be kept motivated, and not bored and exhausted. To achieve this, you apply the *Matron Role* pattern. The matron will take note of all the upcoming celebrations and entertainment opportunities arranging them accordingly, so the team has some fun along with work.

To extra motivate the team and keep them energetic and competitive, you apply the *Compensate Success* pattern. Compensating outstanding performances or groups creates extra motivation in the team, which leads to more successful projects.

To achieve good results and better quality, you may apply the *Develop In Pairs* pattern. A pair can produce more than the sum of what is produced by two individuals. This also helps in keeping an overview of the work. Developers mostly cannot see their own mistakes, but are good criticizers or bug finders in someone else's code.

The ultimate goal of the project is to achieve a successful final product. To ensure this, customer requirements should be discovered. To capture all the scenarios the system will deal with, you can apply the *Scenarios Define Problem* pattern. The product should be tested by an internal group and the customer as well, so apply the *Group Validation* pattern. Individuals alone often miss key bugs.

✣✣✣

Establishing a core team is important at the start of every project. Once a core team is established with the *Size The Organization* pattern, new members can be added to the team with the *Phasing It In* pattern or by using the *Apprenticeship* pattern. New hires can be turned into experts through internship program. The organization can apply the *Day Care* pattern to mentor the new staff. With *Day Care* pattern one expert is helping training all the newbies and the rest of the experts are not interrupted.

Organizations often isolate developers from too much distraction and external inputs. To avoid isolation, use the *Firewalls* pattern with the *Engage Customer* and *Gate Keeper* patterns. The *Gate Keeper* pattern used together with the *Firewalls* pattern keeps the balance between the stakeholder and developer interaction, while the *Engage Customer* pattern complements these two patterns.

The *Develop In Pairs* pattern significantly helps in reviewing the code established by the *Group Validation* pattern. The *Develop In Pairs* pattern helps you make sure that the project advances in accord with the *Someone Always Makes Progress* pattern. Overall, this helps in creating a very effective working environment.

3.2 Capturing the Summary Level Pattern Dynamics

Documenting a pattern language standardizes the vocabulary of experts and helps them communicate with each other [Hafiz et al. 2012]. The diagram in Figure 3, we are visually illustrating the solution part of our summary level pattern. Patterns are represented by states. Transitions between the states are determined by triggers, known also as events, and guards (in square brackets), known also as conditions. Guards must be true in order for the trigger to cause the transition. An empty trigger means the transition is fired immediately.

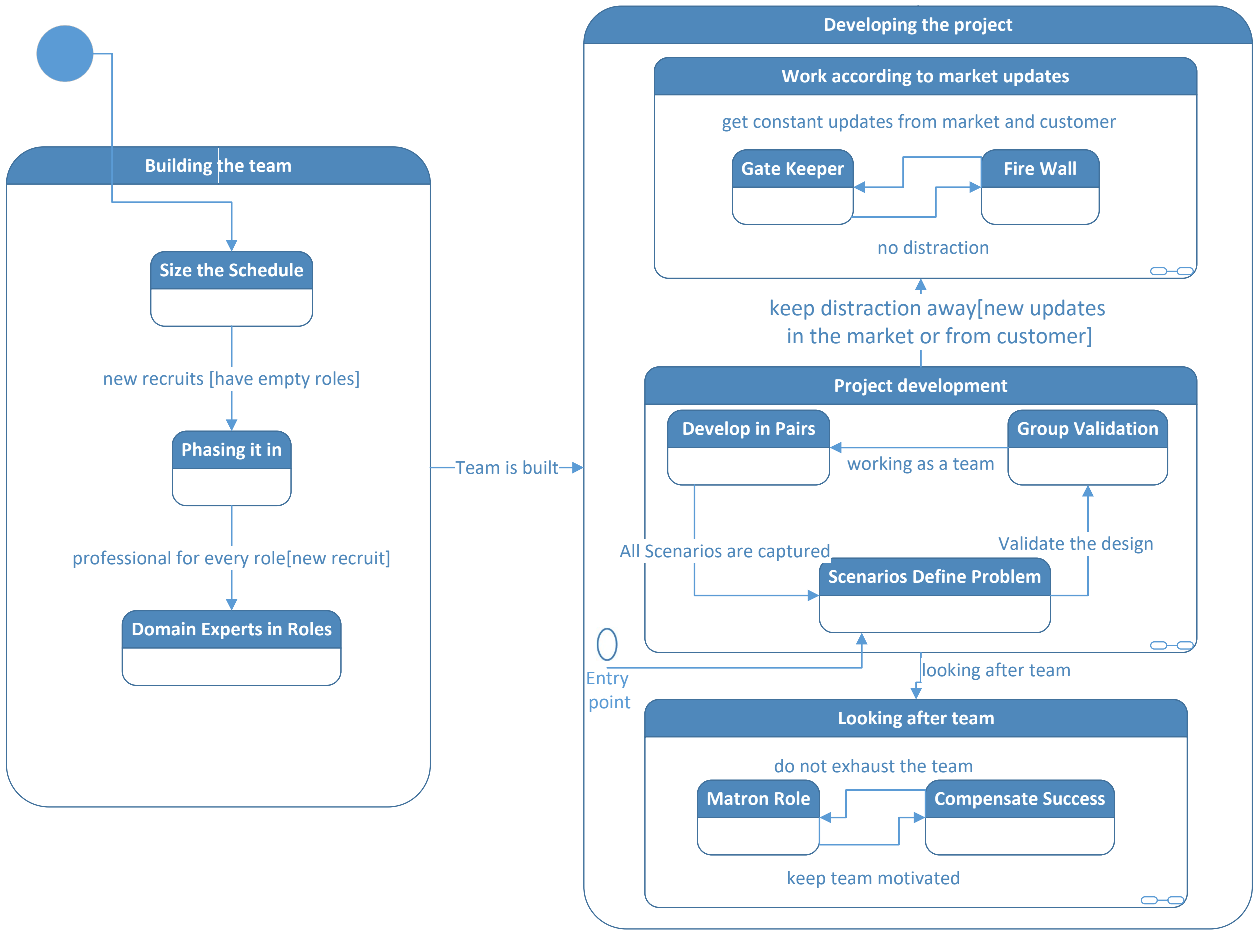

Fig. 3. New Project: a Summary level Organizational pattern

With respect to what we model, triggers represent the decisions by those who apply the pattern sub-language. For example, after applying the *Phasing It In* pattern they may want to have *professionals for every role*, so they fire the corresponding trigger.

Guards represent conditions beyond the control of those who apply the patterns. These may come from the environment, such as when *new recruits* happen after the *Size the Organization* pattern has been applied, or from the roles, such as when *new updates in the market or from customer* occur after the *Project Development* state has been applied.

Composite states are used to combine those patterns that are applied very closely and possibly repeatedly. Composite states help avoid transition explosion. Note that in the *Developing The Project* composite state the starting point is determined by an entry point. In composite states with no entry point, any of the inner states can be the starting point.

4. DISCUSSION

While a pattern story tells a specific event or occurrence, the corresponding summary level pattern is more general and corresponds to many different situations with a matching context. Unlike a summary level pattern, pattern stories are not backed by strong arguments that state why a specific pattern was used. In other words, the discussion of forces is not covered. For example, a part of the pattern story of establishing a new project at one point states the following: "My main job as project coordinator was to put up the *Firewalls* to management until we had our act together." [Coplien and Harrison 2004]. While this sentence refers to a specific pattern (*Firewalls*), it does so in an informal, casual way. In contrast, in our summary level pattern each recommended pattern is discussed, the context is explained, the problem is stated through the discussion of forces, and the solution that will resolve the forces is recommended.

In a summary level pattern, the problem statement is extracted from the pattern story. The need for using each specific pattern in the pattern story is extracted as a force in the summary level pattern. The summary level pattern forces are a compressed form of the forces of the individual patterns it embraces and those forces which connect these patterns.

For example, the main forces in the *Size The Organization* pattern are [Coplien and Harrison 2004]:

—There are limits to the size of software development teams
—Adding people late to the project rarely helps

The forces in the *Phasing It In* pattern are [Coplien and Harrison 2004]:

—You need enough people for critical mass
—You cannot just hire anyone off the street

In *New Project* summary level pattern these forces are compressed.

—There must be an ideal team size based on the project size:
  —Small and sharp teams provide better productivity
  —Software size and complexity demands for an increased team size
—Fill empty roles as early as possible:
  —Adding people late to the project makes project even later
  —All roles must be filled with professionals
  —Not just anyone should be hired

This includes the forces compressed from both *Size The Organization* and *Phasing It In* patterns. The discussion of the forces explain these forces and discuss those forces which bridge the gap between the two patterns.

> Software complexity requires an increased team size, but greater team size significantly decreases its productivity.
> Bringing more staff right at the start will result in a bigger team. Having a bigger team means breaking the project into modules and assigning sub teams to work on these modules. This requires more communication links, which, in turn, decreases productivity.
> There will be empty roles. Either someone leaving the project midway or the task is demanding an additional role.

The discussion of forces are backing each claim of using a specific pattern. The problem part and the discussion of forces part in the summary level pattern debate over the arguments and the conflict of the forces related to individual patterns. It can be said that pattern sublanguage forces compress the forces of the individual patterns it embraces the same way this can be observed with pattern languages [Buschmann et al. 2007].

The solution part recommends the patterns that will resolve the conflicting forces presented in the problem part or in the part on the discussion of forces.

It is important to note that we are not inventing new patterns here. That would go against the very nature of patterns, which are merely being observed and noted by pattern writers. The summary level patterns already exist in pattern stories. The pattern stories that we work with are related to organizational patterns and Coplien and Harrison have provided these at the start of each pattern language [Coplien and Harrison 2004]. These pattern stories have their roots in practical experience and carry the wisdom of applying the patterns in the right order resolving once more the same conflicting forces that accompany the patterns they are about.

5. RELATED WORK

The idea of summary level patterns proposed in this paper is related to Cockburn's summary level use cases [Cockburn 2000]. Summary level use cases interrelate real, user goal level use cases applying the same format for expressing use cases. By this, they provide a big picture of the intended system usage and they do so in a dynamic way, which is something use case diagrams do not achieve.

Buschmann et al. also find pattern languages to be similar to patterns is some aspects [Buschmann et al. 2007]. They speak about societies of patterns: in pattern languages, some patterns appear to aggregate other, "smaller" patterns. The "larger" patterns seems to be generated by the "smaller" patterns they consist of. They observe that this decomposition can appear at several levels. This is similar to our notion of pattern sublanguages, which we see as a "larger" pattern with respect to the "smaller" patterns it contains, Differently than us, Buschmann et al. make no attempt to capture the relationships between the "smaller" patterns within the "larger" ones.

Henney presented pattern stories in a pattern-like style of building up a problem and resolving it in steps [Henney et al. 2005]. He presents pattern sequences for the pattern stories with only a little description of which patterns are to be used. In our approach, the summary level pattern is fully backed by the discussion of forces and direct treatment of the conflicting forces.

A similar approach has been taken by Siddle, who employed pattern sequences to create software architecture [Siddle 2007]. Siddle indicated a possibility of capturing pattern sequences as patterns, but without any further elaboration. In our approach, a pattern sequence can be easily captured by a summary level pattern, since it can be viewed as a special case of a sublanguage.

Furthermore, Siddle used class diagrams to capture pattern sequences. Class diagrams may be sufficient to capture strict orderings, but not the dynamics in pattern sublanguages.

Oberortner presented an interactive pattern story about the design process of the synthetic biology software platform architecture using an activity diagram to capture the pattern story [Oberortner et al. 2012], which is very close to our state diagram approach. However, our state diagram is a diagrammatic view of our summary level pattern.

Nagai et al. [Nagai et al. 2016] present their pattern language for collaborative inquiry as one pattern denoted as *Generator Pattern*. This pattern acts as a summary description of the pattern language. While the intention behind this approach is similar to ours, this summary level pattern presents only the goal level concept of the whole pattern language. In contrast, our summary level pattern involves all the patterns in the underlying pattern sublanguage and their inter-relationship with each other.

6. CONCLUSIONS AND FURTHER WORK

To deal with the complexity of choosing right pattern sequences and understanding pattern languages in general, we propose representing them as patterns. Such summary level patterns can be used to treat meaningful parts of pattern languages: pattern sublanguages. We applied this approach to organizational patterns. Specifically, we expressed the pattern story of establishing a new project [Coplien and Harrison 2004] as the *New Project* pattern. We also captured the dynamics of this pattern by a state diagram.

As it can be observed by comparison, summary level patterns overcome pattern stories in terms of comprehensibility and consistency, with the main contribution being a direct treatment of the conflicting forces.

Our next steps will be to express other organizational pattern sublanguages as summary level patterns and to assess their acceptability by human subjects. In the long run, we intend to explore how this approach can add to organizational pattern comprehensibility along with their animation using text adventure format backed by suggestive language forms [Frt'ala and Vranić 2015].

Acknowledgments

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under grant No. VG 1/0752/14. This contribution/publication is also a partial result of the Research & 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.

We would like to thank our shepherd, Azadeh Alebrahim, for asking tricky questions that made us restate the notions we based our paper on in a more thorough and consistent way. Our sincere thanks go also to the members of the workshop group, namely Michael Weiss, Uwe Van Heesch, Theo Theunissen, Vincenzo Ferme, Elissaveta Gourova, and Thomas Epping for a thorough treatment of our paper.

REFERENCES

Christopher Alexander. 1979. *The Timeless Way of Building*. Oxford University Press.

Christopher Alexander, Sara Ishikawa, Murray Silverstein, Joaquim Romaguera i Ramió, Max Jacobson, and Ingrid Fiksdahl-King. 1977. *A Pattern Language*. Gustavo Gili.

Joseph Blackburn, Michael A Lapré, and Luk N Van Wassenhove. 2006. Brooks' Law Revisited: Improving Software Productivity by Managing Complexity. SSRN, https://ssrn.com/abstract=922768. (2006).

Frederick P Brooks Jr. 1995. The Mythical Man-Month: Essays on Software Engineering. (1995).

Frank Buschmann, Kelvin Henney, and Douglas Schimdt. 2007. *Pattern-Oriented Software Architecture: On Patterns and Pattern Language*. Vol. 5. John Wiley & Sons.

Alistair Cockburn. 2000. *Writing Effective Use Cases*. Addison-Wesley.

James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development*. Prentice-Hall.

Tomáš Frt'ala 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.

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! 2012)*. ACM, New York, NY, USA.

Kevlin Henney, Edwin Schlossberg, and Allan Kelly. 2005. Context Encapsulation—Three Stories, a Language, and Some Sequences. In *In Proceedings of EuroPlop 2005*. Irsee Monastery, Germany.

Marjan Heričko, Aleš Živkovič, and Ivan Rozman. 2008. An Approach to Aptimizing Software Development Team Size. *Inform. Process. Lett.* 108, 3 (2008), 101–106.

Masafumi Nagai, Taichi Isaku, Yuma Akado, and Takashi Iba. 2016. Generator Patterns: A Pattern Language for Collaborative Inquiry. In *Proceedings of 21st European Conference on Pattern Languages of Programs, EuroPLoP'16*. ACM, Irsee Monastery, Germany, 29.

Ernst Oberortner, Douglas Densmore, and J Christopher Anderson. 2012. An Interactive Pattern Story on Designing the Architecture of Clotho. In *Proceedings of 19th Conference on Pattern Languages of Programs, PLoP'12*. ACM.

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.

Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. 2013. *Security Patterns: Integrating Security and Systems Engineering*. Wiley.

James Siddle. 2007. Creating Software Architecture Using Pattern Sequences. In *Proceedings of 12th European Conference on Pattern Languages of Programs, EuroPLoP'07 Workshops*. Irsee Monastery, Germany.

Hironori Washizaki, Masashi Kadoya, Yoshiaki Fukazawa, and Takeshi Kawamura. 2014. Network Analysis for Software Patterns Including Organizational Patterns in Portland Pattern Repository. In *2014 Agile Conference, AGILE 2014*. IEEE, Orlando, Florida, USA.