Is There a Need for a Programming Language Adapted for Implementation of Design Patterns?

RUSLAN BATDALOV, Riga Technical University

The paper contains a summary of a focus group discussion concerning a possibility to design a programming language so that implementation of the known design patterns would be easier than in the existing languages. We discussed whether the patterns community might be interested in such a language, what are expected benefits and problems of this task. The present report summarises the focus group participants' opinions on these matters, expressed in the course of the discussion.

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

ACM Reference Format:
Ruslan Batdalov. 2016. Is there a need for a programming language adapted for implementation of design patterns? *EuroPLoP* (July 2016), 3 pages.
DOI: http://dx.doi.org/10.1145/3011784.3011822

1. INTRODUCTION

Different authors, for example, [Gamma et al. 1995], [Buschmann et al. 2007], [Sommerlad 2007], mentioned the problem of excessive complexity of design patterns implementation. One of the possible ways to reduce this complexity is adding new language-level features that would facilitate design patterns implementation [Batdalov and Nikiforova 2016]. In the focus group, we discussed whether and how a language intentionally designed with this goal in mind might be helpful from the point of view of the patterns community.

2. FORMAT OF THE DISCUSSION

The discussion was held in the 'Six thinking hats' format developed by Edward de Bono [De Bono 2016]. According to de Bono's description, it is a structured group discussion, divided into six consecutive stages. In each stage, all the participants pursue the same way of thinking (metaphorically, wear the same hats) and discuss the problem from a particular angle (process, facts, feelings, creativity, benefits, cautions). After one stage, the whole group switches to the next one, thus ensuring parallel thinking.

During the focus group discussion, the following questions were proposed to the participants:

(1) What problems of design patterns implementation are related to the underlying programming languages? Is there a lack of expressiveness in existing programming languages?
(2) How can a specifically designed programming language be useful?
(3) What opportunities the patterns community would like to see in such a language?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-4074-8/2016/07...$15.00
DOI: http://dx.doi.org/10.1145/3011784.3011822

In order to support the discussion and make it more concrete, the 'Default implementation' feature, described in [Batdalov and Nikiforova 2016], was used as an example. Its idea is to allow to create instances of abstract classes using the same instantiation operator that is used for concrete classes (`new` or its equivalents in other languages) by means of binding abstract classes to their default implementation (defined either at compile-time or at run-time).

The following two sections summarise the opinions expressed by the participants of the discussion.

3. POTENTIAL BENEFITS OF THE APPROACH

The participants agreed that integration of design patterns ideas into programming languages is in line with the current language evolution trends. A well-known example of such evolution is the Iterator pattern, which was described in [Gamma et al. 1995] and later led to introduction of such language-level features as for-each loop and LINQ-expressions [Bishop 2008].

The discussion showed that two types of complexity should be distinguished when we are talking about the design patterns implementation. Partly, this complexity is inherent and inevitable due to additional flexibility given by the design patterns. This type of complexity does not depend on the underlying language. However, in many cases, the implementation complexity is unnecessary and caused by insufficient expressiveness of the existing languages. In such cases, we can expect that a language having features directly corresponding to the ideas and concepts used in design patterns will allow to implement these patterns easier.

The particular areas where we can expect benefits from the discussed approach include the following:

—Big amounts of boilerplate code that have to be repeated on every usage of a pattern and cannot be componentised due to language restrictions (or can be componentised in principle, but this operation does not reduce the complexity of the code).

—Situations when a language feature is only a special case of a general concept, and any behaviour that does not fall under this special case has to be implemented manually.

—Situations when a language lacks features existing in other languages, but cannot easily borrow them due to restrictive design assumptions of this language.

4. KNOWN DRAWBACKS OF THE APPROACH

The participants mentioned also a number of potential problems related to the discussion topic:

—In many cases, reduction of implementation complexity is in fact only hiding this complexity to the level of compiler or run-time environment. Although such complexity hiding can be observed throughout the whole history of programming languages evolution, it may mean that the benefits of introducing new features are not as big as expected.

—Dynamic languages do not seem to gain much from this approach. It is a consequence of the well-known fact that design patterns often solve problems that simply do not exist in dynamic languages because the opportunities to componentise and re-implement behaviour provided by them are generally much greater than in static languages.

—Many design patterns are rather informal (or, in words by [Buschmann et al. 2007], not generic, but *generative*) and allow variety of interpretations. Language-level features derived from such patterns may be hard to design because they should support different usages of the underlying patterns.

—If we facilitate application of design patterns, it may aggravate the common problem of patterns overuse.

—Having a lot of features for all possible design patterns use cases may slow down learning a language and decrease understandability of the code written in it.

—There are other ways to reduce complexity of design patterns implementation, such as componentisation in libraries and frameworks, automatic template implementation, creation of domain-specific languages.

These concerns do not mean that the goal of having a language adapted for implementation of design patterns is senseless or unfeasible, but they have to be taken into account in order to avoid design pitfalls.

5. CONCLUSION

The focus group discussion acknowledged that the problem of excessive complexity of design patterns implementation is still topical and is caused, among other things, by the limitations of the existing programming languages. The idea of designing a programming language adapted for facilitating design patterns usage is promising, but it is related to a number of potential problems. Probably, the final answer about viability of this approach may be given only by an attempt to actually implement such a language.

ACKNOWLEDGMENTS

I want to thank the focus group participants Bogdana Botez, Christian Kreiner, Michael Krisper and Shailaja Shirwaikar for participating and sharing their thoughts, insights, and experience. I am also grateful to Frank Frey for 'mini-shepherding' the focus group proposal, to Ekaterina Andryushchenko for her help in preparation of the discussion and the present report, and to Oksana Nikiforova for the overall supervision of the research.

REFERENCES

Ruslan Batdalov and Oksana Nikiforova. 2016. Towards Easier Implementation of Design Patterns. In *Proceedings of ICSEA 2016, The Eleventh International Conference on Software Engineering Advances*. IARIA, 123–128.

Judith Bishop. 2008. *C# 3.0 Design Patterns*. O'Reilly Media.

Frank Buschmann, Kevin Henney, and Douglas C. Schmidt. 2007. *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*. Wiley.

Edward De Bono. 2016. *Six Thinking Hats*. Penguin Books, Limited.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company.

Peter Sommerlad. 2007. Design patterns are bad for software design. *IEEE Software* 24, 4 (2007), 68–71. DOI:http://dx.doi.org/10.1109/ms.2007.116