Analysing the Need for Training in Program Design Patterns

An empirical exploration of two social worlds

Viggo Holmstedt
University College of South-East Norway
School of Business, Department of Business and IT
Horten, Norway
email: vh@usn.no

Shegaw A. Mengiste
University College of South-East Norway
School of Business, Department of Business and IT
Horten, Norway
email: sme@usn.no

Abstract- **This paper addresses the implications of design patterns knowledge in the social worlds of practitioners and managers from the context of Norwegian software companies. Although there are diverse perspectives on the role and importance of design patterns for object-oriented systems, many academic institutions believe in their relevance, particularly in improving software quality through reusable design. However, when invoking the topic of the relevance of Design Patterns (DP) in a software development process, the engagement varies from no interest to enthusiasm. It was this diverse perspective on the relevance of design patterns that drive us to explore this topic. The paper analyzed practitioners and managers perspectives and our findings revealed a convincing evidence for practitioners' confidence in pattern knowledge and its positive influence on their coding abilities. Our findings are relevant to software design and production, as it addresses methodological issues in software development.**

Keywords-design patterns; object oriented system;practitioner; perspective; manager.

I. INTRODUCTION

The success rate of global systems development was 29% in the year 2015 [41]. Such low rate of success indicates that systems development is a complex process and needs to be addressed with proper planning and guiding. In systems development, earlier design decisions can have a significant impact on software quality; they can also be the most costly to revoke [1]. Design Patterns (hereafter DP, used in plural form) constitute an important tool for improving software quality by providing reusable solutions for recurring design problems.

Design patterns are best practices of specifying and allocating responsibilities to program elements, like classes, packages and components. DP also support the construction of mechanisms based on patterns of class cooperation. Industrial usage and success over a long time typically establishes and confirms a specific design pattern, accepted as a guide to construct mechanisms in complicated systems development contexts.

As Shlezinger et al. [2] indicated, design patterns have over the years provided solutions to design problems with the goal of assuring reusable and maintainable solutions. As a result, DP now exist for a wide range of software development topics, from process patterns to code pattern at various levels of abstraction to maintenance patterns [3]. In the context of object-oriented programming, design patterns are used as building blocks of the architecture and to allow change to evolve in a way that prevents an erosion of the software design [4]. From a software implementation perspective, the value of a design pattern comes from the codification of its specification [5-6]. Regarding usage of DP, Subburaj [13] described the importance of aspects of searching, finding and applying specific patterns, and also convey how an incorrectly applied pattern poses disadvantages.

DP also transfer industrial experience about performing creation and allocating behavior to the internals of classes [7]. Separation of concerns, as between data, logic and presentation, is a success condition in almost all types of systems development [8-11]. Naming is an important characteristic of DP, enabling precise communication and query based search [12]. DP must be constructed and instantiated by developers with experience and ability to realize abstractions with success, including creating and customizing the DP instances.

In terms of usability of DP, a research conducted by Manolescu [14] also indicates that only half of the developers and architects in a software organization tend to use design patterns. The cost of finding and proving the right pattern for a specific mechanism can simply be too high. Despite the fact that there are successful and durable industrial experiences in using DP, as Subburaj [13] clearly noted, DP could be applied in wrong instances and contexts. This alone is a good reason to discuss possible impacts of DP [15], and the importance of training DP skills and knowledge.

Subburaj (ibid) refers to Rising [16], for a debate on formal DP training. Much work is done to construct and establish searchable libraries of DP, reducing the need for formal training. But, pattern catalogs have become too abstract to use for untrained practitioners. We assume that the formal training of classical DP and GRASP (General Responsibility Assignment Software Patterns), which is a methodological approach to learning basic object design [5], would give the practitioner necessary background to assess new in-house patterns, utilize pattern catalogs and correctly instantiate patterns from the practitioner's own knowledge base whenever needed. Formal training would reduce the impact of the abstraction level of pattern catalogs. The debate on the merits of formal training is minimal, and, in this paper, we would like to contribute to this research void.

To meet the huge challenges reported on the usability of DP in practice, academic institutions like our own are offering courses in DP for Object Oriented Systems. Campus students are often impressed by the relevance DP have to their system problems and solutions. Out of campus, we sometimes initiate informal talks with IT directors, developers, managers and other industry practitioners. When invoking the topic of the relevance of DP in a software development process, the engagement varies from no interest to enthusiasm. These informally observed opposites gave us motivation to explore what our own DP students have experienced after leaving school, and after having practiced for a while. We also approached IT employers and other relevant stakeholders without formal training on DP, to have their perception of the importance and relevance of trained DP developers in their respective companies. We acknowledge that many other researchers have investigated the power of DP training to improve the software produced under pattern rules. We appreciate the works of Khomh [15] and Wydaeghe [17] who study and evaluate DP quality attributes. The bottom-line for our investigation is to assess the value or relevance of DP to help software developers to produce better software by guiding them in code production. This will help in assessing the different perspectives on the relevance of running courses in DP, particularly in terms of the experience and minds of the social worlds of practitioners (software developers) and their employers (IT managers and other staff members). To address this research problem, we formulated the following research questions:

Q1: How, when and why do DP trained practitioners perceive relevance of DP knowledge?

Q2: How mutual is practitioners' and managers' understanding of the relevance of DP?

It is our conviction that by answering these research questions, we can contribute to the ongoing research debate between research of DP as a tool to improve software versus DP as a tool to improve thinking and the quality of the practitioner.

The paper is organized as follows: Section II provides an overview of the theoretical framework; and section III presents the research approach and methods, while section IV presents the findings. The last section presents analysis, discussion, and concluding remarks.

II. CONCEPTUAL FRAMEWORK : THE SOCIAL WORLDS FRAMEWORK

The social worlds framework is an analytical framework that has been used in many Science and Technology Studies (STS) [19], and has its roots in the American sociological tradition of symbolic interactionism. The framework focuses on meaning-making among groups of actors- collectives of various sorts – and on collective action – people doing things together and working with shared objects [19]. Strauss [19] citing Shibutani [20] noted that each social world is an arena in which there is some kind of organization; and each social world is a cultural area, where its boundaries are set neither by territory or formal membership but only by the limits of effective communication. The social worlds perspective, as such, conceptualizes organizations "…as being mutually constituted and constituting the systemic order of organizational actions and interactions kept together by individuals and groups commitment to organizational life and work [22]. The notion of groups in this description involves all collective actors (be it a formal organization or group of people) committed to act and interact within the specific social world [23]. In the social world, various issues are debated, negotiated, fought out, forced and manipulated by representatives of the participating social worlds [20].

Huysman & Elkjær [23] argued that organizations could be viewed as arenas where members of different social worlds take different positions, seek different ends, engage in contest and make or break alliances in order to do things they wish to do (ibid, p.8). Over time, social worlds typically segment into multiple worlds (sub-worlds), intersect with other worlds with which they share substantive/topical interests and commitments, and merge [19].

The social worlds perspective has also introduced the notion of agency as well as tension and conflict as triggers for learning among actors in different social worlds [23][25]. Agency is used to denote "various organizational actions and learning and how these are enacted by different kinds of agencies" [23]. Tension and conflict are results of different commitments to different interests, practices and values.

In the context of the study, we adopted the social world perspective as our theoretical framework. We identified two important social worlds: the social world of software developers (practitioners), and the social world of managers (practitioners' superiors). The agencies of both worlds are the production of software, including the learning of best practices to enhance the return on investments.

III. RESEARCH APPROACH AND METHODS

A. *Research Approach*

Our research approach is informed by the priciples of engaged scholarship which advocates a participative form of research to get the perspectives of key stakeholders to understand a complex social problem [25]. One of the main forms of the engaged scholarship research approach is the informed basic research. In this form of research, the researcher acts as a detached outsider of the social system being examined, but solicits advice and feedback from key stakeholders [25][26]. We adopted the informed basic research mainly as our role is detached outsiders, but also we wanted some of our informants in formulating the questionnaire. We already have prepared some grounding by educating a little more than half of the informants through the years 2000 to 2015. Having run DP courses those years, we trusted the benefits to be solid. However, in the research context, that would be like a research lab generated bias, as opposed to the Van de Ven's interactional view. In his view, both the professional and research practices contribute to their common growth of knowledge.

B. Data Collection Methods

We collected data from 28 informants (20 practitioners and 8 IT managers). Both groups contain former students and external contacts. The reason for having some of the former students in the managers' stakeholder group was to assure that most respondents should have at least some knowledge of DP. Van de Ven raises the important question "… why organizational professionals and executives want to participate in informed basic research" [26]. We held this question as an important factor in selecting our respondents. As such we approached only managers with some knowledge of DP, to ascertain their motivation to take part in our investigation.

Our second group of respondents is composed of former students who are now working as system developers in organizations in Norway. Getting the contact information was a challenge since our institution lacks a mechanism to trace former students. So, we relied on technologies like LinkedIn, acquaintances, and a data tool constructed for the purpose. We located about 110 of our former students from courses on DP. We also got a list of about 60 externally collected contacts. Then we used the list and managed to talk to nine of them by phone or face to face, to ascertain that our topic of investigation was relevant to them. During the conversations, we discussed the design of our questionnaire and our chances for having the actual interlocutor as respondent. Those nine helped us in preparing the questionnaires, by giving different comments and sharing their insights.

NVivo [42] is a tool for qualitative research that is specialized for coding and analyzing and for finding trends and interesting opinions. In preparing the questionnaire, we emphasized that all questionnaire items are open to any formulation. This is possible, because the NVivo tool lets us code and analyze the respondents' contributions independent of prior organizing. We let each respondent know that we wanted to learn how, when and why knowledge of design patterns had any importance on his/her professional life after the end of training. We also used each respondent as a possible source of contact information to relevant managers that might have opinions on the relevance of DP.

Finally, we distributed the questionnaire to 170 potential respondents, both managers and practitioners. Out of the 170 emails we sent, we received a total of 28 answered questionnaires that have been analyzed and used in this paper.

The data we got from the 28 respondents has been analyzed using NVivo. As NVivo has huge possibilities for automatic and semi-automatic text analyses, the tool labeled each answer with a code in the place of the full text instruction. The existence of tools for programming the docx format to filter out relevant content from complex structures, available for several programming languages, made this content transformation possible. The transferring of informant documents alleviated the NVivo analysis activities a lot. The filtering of questionnaire content also raised the analysis quality by assuring the non-existence of irrelevant text in the sources.

IV. FINDINGS

In this section, we present our findings. As our focus was to know the perspectives and views of the two social worlds (practitioners and managers) on the relevance and value of design patters in work settings, we present our findings accordingly for the two social worlds. Then, we make a comparison of our findings in the two stakeholder groups.

A. Relevance of DP from the practitioner's perspective

An important occasion for many people in their professional career is the job interview. Therefore, we asked our respondents to comment on what they really think about the relevance of DP knowledge when they apply for a new job in the IT industry. The typical response we received was that: "I hope and believe that it is mandatory to have a good knowledge of Software Design Patterns (SDP). I think SDP is one of the most important aspects of programming." An interesting finding was the distinction between junior and senior developers that reads as follows. "If you apply for a junior position it might not be that relevant because they wouldn't expect you to have knowledge about design patterns, but if you apply for a medium/senior development position it is very relevant." This evidence relies to a discussion concerning introductory training in DP. Some respondents also indicated that Knowledge of DP did not have quite a lot to say when they got their current job.

When we looked for the informants' general perception of DP, we found good evidence for positive perceptions like: "Whenever I need to work on new features / product development, I use design patterns". We also found typical evidence like "it helps with code structure". An interesting finding from respondents free comments is that: "if I have used a common SDP, it might have been easy to understand what I have coded", and "Also DP makes it easier for my colleagues to understand". More evidence for relevance is "I mostly use patterns to communicate intent behind non-trivial code structures."

We wanted to test the evidence material for any sign of enthusiasm, which we interpret as more than just a notion of relevance. We found formulations that we think conveys enthusiasm: "SDP had a big role in my evolution with object oriented programming", and one referring to training: "It has been a great year for me - From finishing school to this point in time I've become a much better coder and problem solver."

Some of the expressions from our respondents on the use of DP show us when and why DP is being used in work settings:

Practitioner 1: "*Yes, weekly, to solve problems.*"

Practitioner 2: "Yes. I used it every day. The main purpose is to be able to understand the code faster and easier if it needs to be changed later on."

Practitioner 3: "Daily. … . We use it in our own development, but it is also essential to understand different design patterns when debugging other developers code efficiently."

One informant also specify two relevant situations: "Yes, first under the design phase of the project and then in the implementation phase."

Much of the evidence refers to daily use: "Everyday, solving problems or reading code in an architectural way to find or create solutions at the right places keeping the code maintainable." There is also evidence in the context of how often, that add concern of code quality: "Most of the time, usually to handle complex situations that would otherwise result in spaghetti code".

As our findings reveal, most of the practitioners believe that DP usage and knowledge improve their code. Our findings also confirm that the improvement is in fact a distinct purpose for using DP as some of the practitioners use DP in order to improve the way they write the code so that to make it as clear and logical as possible.

Our findings also revealed the relevance of DP as a communication agent. As one of the informants indicated: "The software design patterns knowledge will give some help in having meaningful discussions with partners". DP is relevant as a knowledge framework in some situations, helping participants from both different and same social worlds discuss and elaborate solutions.

We specifically asked for informants' perception of the DP influence on time balance in projects. A typical answer for this group is "projects may take a longer time to finish. But it is usually worth it and may save time later." We summarized our findings in the following table (Table 1).

TABLE I. SUMMARY OF PRACTITIONERS' PERCEPTIONS.

Question	Practitioners' perception
How	• has a big role in practitioner's evolution • is a very important aspect of programming • studying DP has been great • makes much better coders and problem solvers • allows for architectural perspective • keeps code maintainable • is timesaver in the longer run • is a knowledge framework
When	• weekly • daily • needing to change existing code • debugging • applying for a job • applying for medium/senior development position • starting new features and product development
Why	• helps with code structure • easy to understand what is coded • communicate intent behind non-trivial code • solve problems • understand others code quickly • long term code maintainability • using DP is doing it right

B. *Relevance of DP from Managers' perspective*

It was important to have informant practitioners with sufficient knowledge of DP to make the questionnaire relevant. The relevance is for managers to have a stake in development. Again, the communication between management and practitioner profits on a mutual understanding of tools and methodologies. Relevant to this concern is evidence like "I think the application of design patterns are very useful for designing faster and more structured applications". More directly targeted at a mutual understanding between practitioners and managers is the following evidence: "Use SDP to increase effectivity in their daily work and to reuse code or methodology from project to project. "

Evidence also displays the relevance of new hires knowledge of DP as follows: "I think very much. It would help keep the number of code lines down overhaul in an application and in the long run perhaps save money".

Managers believe in the positive influence of DP on code improvement:

Manager 1:"Yes, absolutely."

Manager 2: "Yes, because for other people with the same design pattern knowledge, will make it much easier to understand and thereby perhaps much easier to improve upon later".

Manager 2 also confirms the communication and mutual understanding aspects of using DP as follows:

"*The software design patterns knowledge will give some help in having meaningful discussions with partners". We also found an interesting reflection in "it makes me aware of need for pattern creation to create re-usability and standardization."*

Interestingly we found more strong evidence of positive management perceptions of DP. Manger 3 stated the following:

"I want all employers to be as effective as possible, and in this regard use SDP."

Another informant (manager 4) formulated his perception even stronger: "Extremely important". In our material, the positive perception of DP has strong prevalence before any other alternative.

Managers' concern for DP knowledge and new hires are expressed in attitudes like:

"*My understanding is that this influence them",*

"I imagine it does make them more effective.", and

"This has not been on my criteria list (until now)."

The statement "Very relevant, most employers look for design patterns knowledge" represents the most prominent perception among managers. We also found variants of that statement, like "have a positive attitude to design patterns" and "In the current company it is high interest and positivity for it." Some informants thought company size decides level of interest, and stated accordingly: "In bigger companies where you have 100++ employees there is an interest and maintenance of this at a manager level."

A manager focus relevance like this: "It helps seeing pitfalls that has to be handled in the project." Our findings regarding managers' preceptions towards the relevance of DP are presented in the following table (Table 2).

TABLE II. SUMMARY OF MANAGERS' PERCEPTIONS

Question	Managers' Perception
How	• designing faster and more structured applications • opens for meaningful discussions • better communication between developers • positive outlook on SDP • reuse code or methodology • seeing pitfalls
When	• employing new hires • manager is reminded • daily work
Why	• makes hires more effective • code improvement • keeps the number of code lines down • perhaps saves money • has lower maintenance requirement • makes it easier to improve production software later • increases effectivity • create re-usability and standardization

V. ANALYSIS AND DISCUSSION

This section contains analysis of our findings and how they contribute to answering the research questions we set in the introduction. Our findings pointed out that the social world of IT managers have a mixed interest and knowledge about DP and its relevance towards enhancing software development practices. Our findings also demonstrated that the social world of practitioners had a more common interest towards DP with better engagement and knowledge; and even with good understanding of the positive influence of DP usage.

In our research, we wanted to find out how, when and why DP had relevance to practitioners. We constructed a questionnaire that aimed to reveal if DP had any relevance or not in work settings. As highlighted in our findings, most of the practitioners answered positively on the relevance of knowledge in DP to software development. There was only one feeble evidence on the irrelevance of DP among practitioners.

So, in the following subsections, we analyze and discuss our findings around the two research questions we set in the introduction.

A. How, when and why do DP trained practitioners perceive relevance of DP knowledge

We find that the study of DP has been a great personal satisfaction for some of the informants. They also generally think DP give important aspects of programming activities in themselves. DP infer better coding, keep the code maintainable and even give coders a view into architectural considerations. DP also affects problem-solving abilities, and is a timesaver in the long run. The timesaving aspect is especially important in terms of system change claims, which also answers the "when" question. Using DP also creates a knowledge framework to be used for the facilitation of communication between stakeholders. It can even be used to enhance program understanding through pattern reverse-engineering [40].

The reasons for DP's relevance to practitioners are close to the answers for "how" and "when". The understandability is important in multiple directions, that is when coders shall understand other's code, when other shall understand "my code", and even when the coder shall understand his own code. This aspect is also tied to intentions behind code that are difficult to understand without the DP references. The "why" aspect also reveals practitioner responsibility for future needs, as using DP is considered doing the right thing. The same responsibility is even deeper, as it emphasizes positive effect on long term maintainability. Practitioners who have concerns for long term effects of their work, do actually share managers perspectives and interests like return on investment (ROI), e.g. interest of ROI.

Based on our evidence, we assume that knowledge and use of DP is so advanced, that it infers a reinforced perception of ownership to the work. Even more interesting is whether advanced knowledge improves productivity through enhanced self-esteem, as some of our evidence indicated. Judge and Bono [28] pointed out the relevance of self-esteem for job satisfaction and performance. Pierce and Gardner [29] delve deep into these questions in their review of organization-based self-esteem literature. It is much more difficult to find literature on DP knowledge affecting self-esteem and practitioner productivity.

B. How mutual is practitioners' and managers' understanding of the relevance of DP?

The DP literature argues on the importance of IT managers to have insights, reasoning, and techniques to promote and implement design patterns in order to gain operational efficiency and provide strategic benefit for their IT organization. Learning and organizing DP provide an important step [31]. Fowler also state that developers should adapt design patterns to their own environment, and that implementing a pattern for a particular purpose is one of the best ways to learn about it. Cline [31] noted, as design patterns capture distilled experience, they could be used as a tool to improve communication between software developers and other stakeholders, including less experienced developers and managers. Moudam et al. [12] also referred to DP as a communication agent.

Our findings actually highlighted DP as a communication tool to facilitate the interaction between the social worlds of managers and practitioners. The social worlds of managers and practitioners are different, but importantly influenced by the limits of effective mutual communication. Generally, the communication between management and practitioners profits on a mutual understanding of tools and methodology. The hierarchical positions of each member makes the mutual understanding of methods and tools a critical factor. We wanted to gather evidence of the practitioner's perception of the management's and industry's general understanding and attitudes towards DP. Since managers have the model power [33, 34], their knowledge of DP is critical to the practitioners' access to DP and in creating mutual understanding between the two worlds. DP can create mutual understanding by providing a standard vocabulary among practitioners and managers. Under such circumstances, we assume that practitioners who want to use DP will suffer from a weak mutuality of DP understanding and interest.

Even more problematic are the possibilities of misunderstandings and errors induced by different understanding. Literature on this topic for other disciplines exists for example in Hantho et al. [33]. Much closer to our research is Margaret [34] who reports a study of the communication between systems analysts and clients to create requirements. Marne [35] actually construct a DP library as a communication tool. We differ from this by focusing the importance of communication between practitioners and their superiors. DP is by evidence depicted as a tool of communication between individuals of both our social worlds.

We have evidence that most managers have little knowledge of DP, but still express considerable confidence in their employees' usage of them. Managers naturally possess an economical mind-set. The practitioners' more technical mind-set actually has some commonalities with their superiors, which support the mutual understanding between the two social worlds. We found evidence of practitioners' concern for ROI, and also manager's concern for building faster applications more effectively. This is evidence of mutual interests, which is likely to infer a shared interest of communication.

Some managers appreciate relevance of DP based on their assumptions of faster and more structured applications, terms applied to faster designing and development, rather than meaning the application run faster. Still, the interesting part is that the evidence implies an interest for the practitioner's concerns. Besides a general positive appreciation of DP, managers also believe in reuse, enhanced communication between stakeholders in the software process. There is also evidence that managers find DP useful in detecting architectural and technical pitfalls.

There is convincing evidence of mutual understanding between our two social worlds. Even if the mind-sets and perspectives are different, we claim that the reasons mostly fall under the practitioners' concerns as well. DP increases effectivity, and more specifically makes hires more effective. The code improvement, as in lowered number of code lines, is in both stakeholder group's interest. Lower maintenance requirements and easier software improvement are also a concern of practitioners. Reusability is of course also in the practical interest of the coder, while standardization is a general interest of the growing software community that embrace open source solutions. Coherence in DP perceptions between the two social worlds, as well as self-confidence based on knowledge, likely enhance productivity in both social worlds.

Despite the fact that the DP community has been successful in promoting good software engineering practices [37], adoption rates are still low for IT organizations due to lack of discovery and limited education around how to apply design patterns to specific domain contexts [14]. This low adoption rate attributed to the fact that finding DP relevant to a particular problem isn't trivial [14]. This challenge is, in part, due to the nature of how patterns often match a problem domain and each domain needs a distinct approach [37].

When Khom et al. [15] considered criticism to DP; they discussed three GoF [7] examples. Patterns like Flyweight can be a topic of internal discussion, and thus act as social glue among participants. Classical patterns are important not only to infer high software quality, but also to let developers feel at home and find their way in complicated code. Confident and pattern-aware programmers can influence software quality positively, if they are comfortable with the specific pattern instance in use. If the developer finds that a pattern actually decreases the understanding of a software area, it might be because the wrong pattern is used, or that the pattern infers abstractions that decreases both learnability, understandability and the simplicity of debugging. Such abstractions may even amount to emotional resistance [38]. The quality of discussions is better when it is grounded in well-known DP topics, awakening the feel-good of being “at home”, even at work. We would like to promote formal instruction and common knowledge of DP as a social glue between individuals from both the social worlds of practitioners and managers. We would also like to see the construction of meet-up arrangements for the discussion of DP, in order to strengthen the communication attributes of the topic.

Khom et al. [15] states “we consider reusability as the reusability of the piece of code in which a pattern is implemented”. We oppose this, and stated earlier that DP must be constructed and instantiated on site. We find it important to apply the reusability term to the pattern and not the pattern instance code. A target for a specific pattern usage is to improve understandability. A misplaced or miscoded pattern might be the cause of reducing understandability. Internalized pattern knowledge can lead to creative solutions, as opposed to solutions searched by catalogue. The catalogue solutions have to be tested and found usable for the specific problem, and are often only partially understood. We therefore promote formal instruction of DP to internalize a small set of pattern repertoire in the minds of the social worlds’ individuals.

VI. CONCLUSION

In this paper, we empirically assessed whether or not knowledge of DP is relevant for managers and practitioners in software development companies. Our findings revealed that there are differences between the social worlds of managers and practitioners in how they perceive DP as a vehicle to enhance performance of development teams. Practitioners expressed high level of relevance for the knowledge of DP, while managers put a lower level of relevance to DP. However, our findings also revealed that both social worlds believed in DP’s ability to act as a communication tool, and that the quality of mutuality in DP perceptions between the two social worlds is good.

Several works focus on measurable characteristics by inspecting collections of code. Hegedus et. al. [39] inspect the quantitative grounding for evaluating the effect of DP on software maintainability. In contrast, we wanted to investigate the effect on human thinking and courage in software building. Meaningful naming of components, like classes, fields and enum type items, is an example of less measurable code characteristics that still may have huge effect on software maintainability, because of its ability to guide human understanding.

We do not focus DP’s characteristics as a direct software improving tool, but indirect as a human helper. DP help humans think, and thereby help humans improve design and program code. Tahvildari et al. [40] focus on their own classification schemas to help designers understand relationships between patterns, but do not connect DP directly to the assistance to think, or to the user’s attitudes towards DP. Even if the correct usage of DP reduces risk, the adoption rates are still low. Manolescu [14] claimed low discovery and education to be important factors, which makes it interesting to investigate the present attitudes of employers and former students of DP, and detect any importance for their professional life. If practitioners’ knowledge and usage of DP enhance them as coders, it is of great significance when their managers find positive relevance in their usage of DP. The practitioners will feel support and encouragement to continue their good work.

Measuring attributes of software created with patterns would oversee all the varying ways of instantiation, all the varying machine and OS variants and escape future changes in OS/machine dependencies. We therefore and alternatively suggest the discussion of practitioners’ self-confidence and its effect on productivity, a coherence that can prove to be more future-proof, being grounded in the human nature.

REFERENCES

[1] E. Folmer. and J. Bosch “A pattern framework for software quality assessment and tradeoff analysis.” International Journal of Software Engineering and Knowledge Engineering, Vol.17, no. 04, pp.515-538, 2007.

[2] G. Shlezinger, I. Reinhartz-Berger, and D. Dori. “Modeling design patterns for semi-automatic reuse in system design. Cross-Disciplinary Models and Applications of Database Management: Advancing Approaches.” Advancing Approaches, pp.29, 2011.

[3] S. Henninger. and V. Corrêa. “Software pattern communities: Current practices and challenges.” In Proceedings of the 14th Conference on Pattern Languages of Programs, ACM,, 2007.

[4] D. J. Ram and M.S. Rajasree. “Enabling Design Evolution in Software through Pattern Oriented Approach, in Object-Oriented Information Systems” In Proceedings of 9th INternational Conference, OOIS 2003, Geneva, Switzerland, PP. 179- 190, September 2003.

[5] C. Larman. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. 2005: Pearson Education India.

[6] L. Ackerman and C. Gonzalez. “The value of pattern implementations.” DR DOBBS JOURNAL, Vol. 23, no. 6, pp. 28-34, 2007.

[7] J. Vlissides, et al. “Design patterns: Elements of reusable object-oriented software.” Reading: Addison-Wesley, Vol. 49, no. 120, pp. 11, 1995..

[8] P. Tarret al. «N degrees of separation: multi-dimensional separation of concerns.” In Proceedings of the 21st international conference on Software engineering. 1999. ACM.

[9] V. Kulkarni and S. Reddy “Separation of concerns in model-driven development.” IEEE software, Vol. 20, no. 5. Pp. 64- 69, 2003.

[10] M. Aksit, B. Tekinerdogan, and L. Bergmans. Achieving adaptability through separation and composition of concerns. 1997.

[11] T. Mens and M. Wermelinger “Separation of concerns for software evolution.” Journal of software maintenance and evolution: research and practice, vol. 14, no. 5, pp. 311-315, 2002.

[12] Z. Moudam and N. Chenfour “Design Pattern Support System: Help Making Decision in the Choice of Appropriate Pattern.” Procedia Technology, Vol. 4, pp. 355-359, 2012.

[13] R. Subburaj, G. Jekese, and C. Hwata “Impact of Object Oriented Design Patterns on Software Development.” International Journal of Scientific & Engineering Research, vol. 6, no. 2, pp. 961-967, 2015.

[14] D. Manolescu et al. “The growing divide in the patterns world.” Software, IEEE, vol. 24, no. 4. Pp. 61-67, 2007.

[15] F. Khomh and Y. G. Guéhéneuc. “Do design patterns impact software quality positively?” In Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. 2008. IEEE.

[16] L. Rising. “The Benefit of Patterns.” IEEE software, vol. 27, no. 5, pp. 15-17, 2010.

[17] B. Wydaeghe et al. «Building an OMT-editor using design patterns: An experience report.” In Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings. 1998. IEEE.

[18] A. E. Clarke. and S.L. Star “The social worlds framework: A theory/methods package.” The Handbook of Science & Technology Studies, vol. 3, pp. 113-137, 2008.

[19] A. L. Strauss. Scientists and the evolution of policy arenas: The case of AIDS., in stone symposium of the society for the study of symbolic interaction. 1991: San Fransico, CA.

[20] T. Shibutani. “Reference Groups as Perspectives.” American Journal of Sociology, vol. 60, no. 6, pp. 562-569, 1955.

[21] B. Elkjær. “Organizational learning the ‘third way’.”Management learning, vol. 35, no. 4, pp. 419-434, 2004.

[22] B. Elkjærand M. Huysman “Social worlds theory and the power of tension.” IN: D., Barry & H., Hansen (Eds.), The SAGE handbook of new approaches in management and organization, pp. 170-177, 2008.

[23] M. Huysman and B. Elkjær. “Organizations as arenas of social worlds: Towards an alternative perspective on organizational learning?” In Organizational Learning and Knowledge Capabilities Conference. 2006.

[24] A. E. Clarke.. Social organization and social process: Essays in honor of Anselm Strauss, 1991.

[25] A. H. Van de Ven Engaged scholarship : a guide for organizational and social research. 2007, Oxford ; New York: Oxford University Press. xii, 330 p.

[26] A. H. Van de Ven Engaged scholarship a guide for organizational and social research. 2007, Oxford University Press: Oxford ; New York. p. 1 online resource.

[27] A. Alnusair T. Zhao, and G. Yan “Rule-based detection of design patterns in program code.” International Journal on Software Tools for Technology Transfer, vol. 16, pp. 315-334, 2014.

[28] T. A. Judge and J.E. Bono. “Relationship of core self-evaluations traits—self-esteem, generalized self-efficacy, locus of control, and emotional stability—with job satisfaction and job performance: A meta-analysis.” Journal of Applied Psychology, vol. 86, pp. 80-92, 2001.

[29] J. L. Pierce and D.G. Gardner. “Self-Esteem Within the Work and Organizational Context: A Review of the Organization-Based Self-Esteem Literature.” Journal of Management, vol. 30, no. 5, pp. 591-622, 2004.

[30] D. Alur et al. Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies. 2003: Sun Microsystems, Inc.

[31] M. P. Cline. “The pros and cons of adopting and applying design patterns in the real world.” Communications of the ACM, vol. 39, no. 10, pp. 47-49, 1996.

[32] A. M. Kanstrup and E. Christiansen. Model power: still an issue?, in Proceedings of the 4th decennial conference on Critical computing: between sense and sensibility, pp. 165-168, 2005, ACM: Aarhus, Denmark.

[33] A. Hantho, L. Jensen, and K. Malterud. “Mutual understanding: a communication model for general practice. “Scandinavian Journal of Primary Health Care, vol. 20, no. 4, pp. 244-251, 2002.

[34] T. Margaret. “Establishing Mutual Understanding in Systems Design: An Empirical Study.” Journal of Management Information Systems, vol. 10, no. 4, pp. 159-182, 1994.

[35] B. Marne et al. “A Design Pattern Library for Mutual Understanding and Cooperation in Serious Game Design, in Intelligent Tutoring Systems”. 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012. Proceedings, S.A. Cerri, et al., Editors., pp. 135-140, 2012, Springer Berlin Heidelberg: Berlin, Heidelberg.

[36] F. Buschmann, K. Henney, and D. Schimdt. Pattern Oriented Software Architecture, vol. 5, 2007: John Wiley & Sons.

[37] S. J. Bleistein et al. Linking requirements goal modeling techniques to strategic e-business patterns and best practice. in 8th Australian Workshop on Requirements Engineering (AWRE’03). 2003. Citeseer.

[38] V. Holmstedt and S. A. Mengiste. «Effect of Code Maintenance on Confidence in introductory object oriented programming Courses.” IN: IRIS2016. 2016: Sweden. Unpublished

[39] P. Hegedűs et al.. Myth or reality? analyzing the effect of design patterns on software maintainability, in Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity, pp. 138-145, 2012, Springer.

[40] L. Tahvildari and K. Kontogiannis. «On the role of design patterns in quality-driven re-engineering. in Software Maintenance and Reengineering.” IN: Proceedingsof Sixth European Conference on. 2002. IEEE.

[41] Standish Group 2015 Chaos Report, available at: https://www.infoq.com/articles/standish-chaos-2015 [accessed March 2017]

[42] http://www.qsrinternational.com/what-is-nvivo [accessed March 2017]