The Journal of Systems and Software 140 (2018) 74–90

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Measuring object-oriented design principles: The results of focus group-based research

Johannes Bräuer[a], Reinhold Plösch[a,*], Matthias Saft[b], Christian Körner[b]

[a] Johannes Kepler University, Department of Business Informatics - Software Engineering, Altenbergerstraße 69, Linz 4040, Austria
[b] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, Munich 81739, Germany

ARTICLE INFO

Article history:
Received 24 October 2017
Revised 16 January 2018
Accepted 1 March 2018
Available online 3 March 2018

Keywords:
Design best practices
Design rules
Design principles
Software design quality
Design improvement

ABSTRACT

Object-oriented design principles are fundamental concepts that carry important design knowledge and foster the development of software-intensive systems with a focus on good design quality. They emerged after the first steps in the field of object-oriented programming and the recognition of best practices in using this programming paradigm to build maintainable software. Although design principles are known by software developers, it is difficult to apply them in practice without concrete rules to follow. We recognized this gap and systematically derived design best practices for a number of design principles and provide tool support for automatic measurement of these practices. The aim of this paper is to examine the relationship between design best practices and 10 selected design principles. This should provide evidence whether the key design aspects of the design principles are covered. We conducted focus group research with six focus groups and 31 participants in total. In parallel, each group discussed five design principles and assessed the coverage by using the Delphi method. Despite suggestions of additional design practices that were added by the participants, the result reveals the impact of each design best practice to the design principle and shows that the main design aspects of the design principles are covered by our approach and is therefore feasible to derive concrete design improvement actions.

1. Introduction

How does a good software design stand out from others? One answer could be that it must be obvious that someone has taken care of it (c.f., Feathers in Martin, 2008). The object-oriented programming paradigm and the accompanying languages have placed much responsibility on software developers regarding the design of a software system. For example, the ways of slicing abstractions, using encapsulation, and providing appropriate interfaces mainly depend on the professionality of the programmer and designer. Consequently, each software developer takes responsibly for designing a software solution whose design is simple and orderly. In short, a software developer must take care of the resulting software design.

Essential instruments for carrying out this task are design principles since they support a software developer in developing a software system that takes full advantage of the object-oriented programming paradigm. In this work, we do not focus on general design principles like coupling or cohesion, but we concentrate on more specific design principles like the single responsibility principle or the open-closed principle. In general, this kind of design principles define guidelines to ensure design quality aspects (e.g., maintainability, portability, or functional suitability) and avoid the introduction of traps and pitfalls (Dooley, 2011). Martin (2003), for instance, broadly discusses a set of object-oriented design principles that are also used to teach software design aspects and assess the quality of software systems (Samarthyam et al., 2013). An example of this set of design principles is the single responsibility principle, which states that every abstraction should have only one responsibility expressed by the reasons to be changed. In other words, when a class has two reasons to be changed and the changes are not related to each other, it is likely that the class contains too much responsibility (Martin, 2003). Consequently, in such cases, each responsibility should be separated into a single class. Adherence to this guideline eases the maintainability and readability of the source code and reduces the likelihood of side effects. In general, it enhances the design of the software.

To reveal the importance of various design principles in practice, we surveyed 104 software engineers and architects (Plösch et al., 2016a). In this survey, we identified that the above-mentioned single responsibility principle, as well as the separation of concern principle and the information hiding principle, are the top three principles known by the participants of the survey. Al-

* Corresponding author.
E-mail address: reinhold.ploesch@jku.at (R. Plösch).

https://doi.org/10.1016/j.jss.2018.03.002
0164-1212/

though engineers and architects are aware of these principles, they noted – and we observed – that there are difficulties in following them while developing or designing a software system. One of the reasons is that the descriptions of design principles are still too vague to be easily understood and correctly implemented in the design. Based on this shortcoming, we analyzed the design principles to systematically identify related design best practices (aka design heuristics according to Riel (1996)). To determine the violations of these practices directly in the source code, we built a measurement tool as introduced in Plösch et al. (2016b). Finally, we defined a Quamoco-based quality model that assigns these practices to the principles in a formal way (Plösch et al., 2016a).

For the validation of this design quality model, we followed a bottom-up strategy and started examining the design best practices. Therefore, we conducted a broad survey of 214 participants and derived findings regarding the importance of different design best practices on design quality in general (Bräuer et al., 2017b). In this survey, design principles were ignored.

Based on the conducted surveys, we have gained evidence on the importance of these design knowledge-carrying concepts (design best practices and design principles) and the findings met our expectations. However, the important and remaining question is whether the prosed design best practices for a specific design principle cover the essential aspects of the principle or just touch on some minor design aspects. To answer this general question, we derived following research questions:

- RQ1: How important are design best practices concerning their assigned design principle?
- RQ2: Are there additional design best practices for operationalizing a design principle?
- RQ3: To what extent can the design knowledge of a design principle be grasped by the associated design best practices?

RQ1 focuses on the importance of a best practice in relation to assigned principles. This should reveal those best practices that stronger impact a design principle compared to others. In course of discussing this relationship, it can be assumed that participants identify aspects that are not addressed by our proposed set of practices. This is why RQ2 explicitly addresses this point and collects those ideas that have been unconsidered by us. Finally, it is necessary to understand to what extent the proposed design best practices cover the design principles. By answering RQ3 in terms of a completeness assessment for each principle, we will gain evidence on whether our set of design best practices can claim to be able to measure design principles and to provide the basis for deriving improvement actions.

To reveal answers to the three research questions, we conducted focus group research with 31 experts in software design. The focus group research approach was selected since it provides a framework for in-depth discussions, place for gathering new ideas, and techniques for working on a research question in a collaborative manner. Latter aspect means that participants of a focus group can learn from each other to get a more elaborated view on the topic in particular on design principles. This helps to enhance the overall validity of the study due to a common understanding of the principles – a circumstance which is difficult to achieve by using, for instance, interviews or surveys that are conducted individually.

The remainder of this paper is structured as follows. The next section provides more details on design principles and the research approach. Section 3 explains the design of this investigation including the discussion process of a focus group and a summary of the primary results at the end. Section 4 draws a conclusion of the research process and discusses the results. Finally, Section 5 highlights the limitations of this investigation before it presents a conclusion and avenue for future work in the last section.

2. Related work

This work examines the research area of measuring object-oriented software design quality by verifying the adherence to the design principles using design best practices. To further examine this topic, the focus group research method is selected. Thus, related work pertaining to the research area and research method is discussed.

2.1. Design principles

Approaches for measuring object-oriented software design have received much attention since Chidamber and Kemerer proposed a well-known metrics suite that measures the properties of object-oriented design (Chidamber and Kemerer, 1994). Some approaches have adapted and extended this suite to understand the characteristics of particular object-oriented design aspects (Subramanyam and Krishnan, 2003; Srivastava and Kumar, 2013), or investigated specific aspects of more general design principles related to cohesion, coupling, and inheritance (Briand et al., 1999, 2001). In addition to measuring, decisions for refactoring can be identified and design improvements can be driven. However, it has been recognized that using single metrics and considering them in isolation does not provide enough insight for proposing sound design improvements (Marinescu, 2004).

Our work focuses on understanding the application of design principles in software design. In this regard, we do not consider design principles at the coarse-grained level of low coupling, high cohesion, moderate complexity, and proper encapsulation, as proposed by Coad and Yourdon (1991) or described by Meyer (1997), but rather on a finer level and with principles addressing a particular design issue, like the single responsibility principle or the open-closed principle.

It has been discussed that these so-called *fine-grained design principles* are more accurate and provide useful guidance for building high-quality software design (Dooley, 2011; Sharma et al., 2015). Further, they are used to organize and arrange the structural components of the (object-oriented) software design, build up a common consensus about design knowledge, and support beginners in avoiding traps and pitfalls. Although fine-grained design principles break down design aspects, however, they are still too abstract to be directly measured.

Except for one work that touches on the operationalization of design principles, we are not aware of any broader investigation in this regard. The work that addresses this research topic to some extent concentrates on design principles related to good interface design (Abdeen et al., 2013). In more detail, the team analyzes the design of interfaces to evaluate the adherence of interfaces to the interface segregation principle and the program to an interface, not an implementation principle. Therefore, they assign cohesion metrics to the principles and measure these metrics on a set of open-source projects. The results of this research show that software developers follow this narrow set of principles (Abdeen et al., 2013), but the work does not provide any guidance on how to improve a design that does not follow the measured principles.

Other work on measuring and assessing design principles is shown by Samarthyam et al. (2013). They manually assess the design of real-world projects by relying on the expertise of design experts. During these assessments, they observe that the cause of poor design quality is the violation of design principles. However, the community is missing an analysis framework that maps design issues to the violations of design principles as highlighted by Samarthyam et al. (2013) and Sharma et al. (2015).

Our previous work already addressed this gap and proposed a model that assigns design best practices to design principles (Plösch et al., 2016a). We consider design best practices to be mea-

surable design properties that can be operationalized by a static code analysis tool (Plösch et al., 2016b). The value of using design best practices, aka design heuristics (Riel, 1996), to link abstract design principles to quantitative software properties is underlined by Churcher et al. (2007). While this group has a strong focus on visualizing the violations of design heuristics, they argue that this measuring technique is a valuable tool to identify design issues and evaluate design quality (Churcher et al., 2007).

To summarize, there are a large number of approaches and studies available that propose approaches to measure more general design principles like abstraction, coupling, cohesion, or inheritance by means of metrics of design smells. However, there is little work available to operationalize more specific design principles like the single responsibility principle or the open closed principle. Those approaches available, concentrate on either metric-based or smell-based approaches for measurement, while our approach relies on design best practices that provide specific hints how to improve the design of the source code. In our previous work (Bräuer et al., 2017b), we showed the importance of our design best practices in general and without considering the aspect of measuring design principles. The focus of this work is to find out about the importance of our design best practices in the context of the design principles and to get insights to which extent the design principles can be measured by our design best practices.

2.2. Focus group research

To address the above mentioned research questions, we choose the focus group research approach. Focus groups are carefully planned discussions with the flexibility to obtain the personal perceptions of the participants in a defined research area (Kontio et al., 2008). We apply this research approach since the research question focuses on a defined research area and focus group members can express their opinions, resulting in insightful information. The flexibility to capture the individual opinions of participants is required because the discussion of software design quality is a complex matter and context-dependent.

Although there is no related investigation of design principles by using focus group discussions, focus group research is an accepted empirical research approach within the software engineering research community (Kontio et al., 2008). One of its strengths is the discovery of new insights due to the interactive nature of the setting and different backgrounds of the participants (Kontio et al., 2008). This supports researchers in getting new ideas that might not have been considered beforehand. Further, focus groups are well suited to obtain feedback on how models or concepts are presented (Edmunds, 2000). For instance, this approach has been successfully used to validate a process framework for embedded systems engineering (Charalampidou et al., 2014).

3. Focus group research design

To answer the research questions mentioned in the Introduction, participants need to understand the concept and characteristics of each design principle. Further, they may have questions about some aspects that need to be clarified; otherwise, design principles could be wrongly interpreted. With a solid understanding of the design principle, it is possible to judge whether design best practices are related to a design principle. According to these circumstances, we decided to conduct focus group research.

While the dynamics of the focus group and relatively small sample size may bias the results of the investigation (Kontio et al., 2008), some of our decisions regarding the research method try to reduce these risks explicitly. The entire design of the research method and its combination with Delphi as a data collection method is aligned with the guidance given by Kontio et al. (2008).

3.1. Research planning

The focus group method is suitable for gathering feedback on new concepts or models as well as for generating ideas (Kontio et al., 2008). We decided to use this approach to discuss a defined set of design principles in software engineering. The selection of this set of principles is based on their practical relevance. Thus, we conducted a survey beforehand to find out the ten most important design principles from an original set of 31 identified object-oriented design principles (Plösch et al., 2016a). Consequently, the subjects of the investigation in this work are the *single responsibility principle (SRP), information hiding (IHI), don't repeat yourself principle (DRY), open closed principle (OCP), acyclic dependency principle (ADP), interface segregation (ISP), favor composition over inheritance principle (FCOI), command query separation principle (CQS), common closure principle (CCP),* and *program to an interface not an implementation (PINI)*. Below, the definition of each principle is provided as used throughout the focus group research.

- **Single responsibility principle:** If a class has two reasons to be changed and the changes are not related to each other, the functionality of the class has to be split into two separate classes (Martin, 2003). When following this principle, each class will handle only one responsibility and extensions will be individually addressed in each class.
- **Information hiding principle:** A class is not allowed to expose design decisions that are characterizing its implementation.
- **Don't repeat yourself:** "Every piece of knowledge must have a single, unambiguous and authoritative representation within a software system" (Hunt and Thomas, 1999, p. 27), that is to say, neither duplicated data structures or source code nor meaningless source code documentation.
- **Open closed principle:** Source code should be written and the design should be implemented in a way that allows adding new functionality with minimum changes in the existing source code.
- **Acyclic dependency principle:** The dependency structure between packages must be a directed acyclic graph meaning that there must not be cycles in the dependency structure on package level (Martin, 2003).
- **Interface segregation principle:** "Classes should not be forced to depend on methods [of an interface] that they do not use" (Martin, 2003, p. 137). Instead of one fat interface, a set of small interfaces is preferred whereas each interface serves a particular group of classes.
- **Favor composition over inheritance:** Object composition instead of class inheritance should be used to reuse functionality.
- **Command query separation:** A method should either modify an object (command) or return data of an object (query). These two concepts should not be mixed.
- **Common closure principle:** Classes within a package should be closed together against the same kind of changes. A change affecting a package therefore affects all classes in that package (Martin, 1996).
- **Program to an interface, not an implementation:** Interfaces or abstract classes should be used instead of concrete classes.

To each of these design principles, we assigned at least one design best practice (aka design rule) from the list of practices in the Appendix A. In most cases, multiple design best practices are linked to a design principle depending on the facet richness of addressed design aspects. Besides, some principles share similar design intentions resulting in multiple assignments of practices. Before conducting this investigation, we systematically evaluated each design best practices to understand their importance for good object-oriented design, using a survey-based approach (Bräuer et al., 2017b). As a result, we became aware of the impor-

tance levels of the design best practices; something we expected. Although we obtained a good understanding of the importance of design best practices for object-oriented design by using this survey, the relation between design best practices and design principles is still poorly understood - a gap this work tries to bridge.

For the sake of clarification, the validation of these design best practices does not only rest upon their name and description because we also developed a measurement tool called MUSE that identifies the non-conformance of these practices (Plösch et al., 2016b). Thus, the tool implements the logic of the design best practices and quantifies violations in source code written in Java, C#, or C++. In previous work, MUSE was applied in the context of different research activities and in real-world projects of industrial partners (Plösch et al., 2016a,b).

Verbally conducting a focus group discussion may be dominated by opinion leaders or the group behavior (Kontio et al., 2008). This is the reason why we defined the first requirement to conduct anonymous discussions in which team members are demanded to contribute anonymously. Further, design principles are complex and difficult to grasp in a short session. Accordingly, we defined as the second requirement that there must be a way for the participants to study design principles independently. Finally, there was the constraint that the participants could not meet in person because traveling costs could not be refunded. Thus, the third requirement focused on conducting the discussions remotely. To address all of these needs, we concluded to design and conduct virtual (online) focus groups as defined and suggested in Turney and Pocknee (2005).

3.2. Focus group design

Throughout designing the focus groups and considering the necessary effort, it was not possible to discuss all ten design principles in each group. Consequently, the design principles were divided into two sets based on pre-defined requirements. First, we tried to keep the number of assigned design best practices and therefore the effort for judging them balanced between the focus groups. For instance, IHI and OCP that both have many assigned practices were kept separated. Second, each set had to contain one of the two package-related design principles that are ADP or CCP. Third, we separated FCOI and PINI since these two design principles address abstraction issues. Finally, we derived the first set containing SRP, IHI, FCOI, CQS, and CCP; and the second set comprising OCP, DRY, ADP, ISP, and PINI.

A typical focus group research approach consists of four to six focus groups with a sample size of four to eight participants (Kontio et al., 2008). Because smaller groups generate high levels of participant involvement (Kontio et al., 2008), we decided to form groups of five participants. Furthermore, we aimed at having at least three groups discussing one set of design principles, resulting in 15 opinions for each design principle. Consequently, it was necessary to acquire at least 30 participants for the six focus groups.

3.2.1. Selection of participants

Before recruiting participants, we defined the minimum software engineering skills needed. Thus, participants must have good to top experience in any of the three object-oriented programming languages: Java, C#, or C++. This requirement reduces the risk that participants are not able to grasp the intention of a design principle including their practical relevance and impact on design quality. For the evaluation of this experience, the participants had to provide a self-appraisal on a five-level scale where the two highest values were good and top.

We acquired participants for our study in three ways. First, we invited people who completed our previous survey about the importance of design best practices. In this survey, participants subscribed to receive a summary of the survey, and in distributing this document, we invited them to participate in a focus group. Although this acquisition represents a threat to validity to the advance in knowledge of re-participating members, we controlled this threat by providing the same (or even more) information of design best practices to all focus group members. Second, we asked local companies with a strong focus on software development to promote the participation as a means for enhancing object-oriented design skills of their software developers. Lastly, an industrial research partner incorporated the focus group discussions as part of a senior developer training. In this training, the design principles play a central role, meaning that these participants are perfect candidates for our focus group research.

3.2.2. Segmentation of participants

The segmentation of focus groups addresses the proper composition of the groups and offers two basic advantages according to Morgan (1997): (1) the segmentation allows building comparable dimensions into the entire research project and (2) it facilitates the discussions by making group members more similar to each other.

Considering these two advantages, we composed the six focus groups depending on different constraints. For instance, some participants were limited in their availability; hence, we assigned them to an earlier or later group discussion. Furthermore, we consolidated the participants recruited from the same company to support the homogeneity of the teams. Briefly summarized, the main characteristics of the groups are shown below:

- **FG-I:** A group of members working in software engineering research departments with a strong academic background. These participants have completed our previous survey. Thus, we can assure that all participants fulfill our requirements regarding their software engineering qualifications.
- **FG-II:** This group contains volunteers recruited via LinkedIn and ResearchGate discussion groups. A background check and their contribution to our previous survey proved their software engineering competencies. However, the group members work in different application domains.
- **FG-III to V:** Our industrial research partner constituted three focus groups. More specifically, these participants are senior software developers enrolled in an in-house key developer training offered by the research partner. Although we could not conduct a background check of the engineering skills as conducted for other participants, our research partner confirmed that the skills are at – and in most cases above – the required level. Further, the training program focuses on teaching design principles and establishing awareness in following them. Therefore, trainees must complete assignments such as principle hunting in their source code as well as principle-based improvement activities. Overall, these trainees are perfect candidates for this research as they have a thorough understanding of design principles and their violations and have – due to their seniority level – perfect software engineering knowledge.
- **FG-VI:** A local industry partner set up a focus group. These members are working in industrial automation and are senior developers with good experience in building embedded systems using object-oriented technologies for more than five years.

3.3. Focus group discussion

Since we decided to constitute online focus group discussions, appropriate tool and data collection methods were required. From the tool perspective, it is recommended to rely on university learning management systems (Turney and Pocknee, 2005). This results

in the benefit that the research process takes advantage of the quality, security, and privacy regulations defined by the university (Turney and Pocknee, 2005). Further, the participant's anonymity and confidentially can be ensured (Turney and Pocknee, 2005). For this research, the Moodle[1] service, which is hosted by the JKU, was used. We have already applied this service to manage coursework and student teams, so no training was needed.

To collect the data from the group discussion, we selected the Delphi method. In the literature, the Delphi method is suggested for this purpose because it structures the discussion process and supports its documentation (Adler and Ziglio, 1996). Linstone and Turoff (1975) provide a more general definition of the Delphi method and state that it structures a group communication process so that the process is effective in allowing participants to deal with a complex problem. Therefore, the implementation of the process must provide feedback on individual contributions, an assessment of the group judgment or view, the opportunity for individuals to revise views and judgments, and some degree of anonymity (Linstone and Turoff, 1975).

Although we could have conducted a traditional survey or interviews during the focus group discussions, we decided to apply the Delphi method. The main benefit of this method is that participants can express their opinion in a first round that can be revised after considering the opinion of the other participants of the focus group. Generally, this is a stronger approach for interrogating participants and helps participants to rethink their first opinion, which may be dominated by their software development duties and the environment they are working. Another key advantage of the method is that it avoids the confrontation of experts (Okoli and Pawlowski, 2004), supporting our first requirement of conducting an anonymous discussion. Lastly, Delphi does not require participants to meet in person (Okoli and Pawlowski, 2004). This perfectly aligns with our third requirement and does not limit the research to local experts.

Before starting the discussion process, we assigned one of the two sets of design principles to each focus group. More specifically, the focus groups FG-I, FG-III, and FG-V analyzed the design principles SRP, IHI, FCOI, CQS, and CCP, while the focus groups FG-II, FG-IV, and FG-VI analyzed the design principles DRY, OCP, ADP, ISP, and PINI. Accordingly, at least 15 experts worked on each design principle.

Based on the process proposed by Okoli and Pawlowski (2004), we divided the discussion of our focus groups into three phases. An overview of these three phases is shown in Fig. 1 highlighting the main activities and artifacts. The first phase was called the *brainstorming phase* and introduced the topic and design principles to the participants. After this step, the *clarification phase* followed. This focused on identifying white-spots and clarifying the importance of design best practices in the context of a given design principle. Due to the result of the ongoing group discussion, the last phase, the *completeness assessment phase*, asked the participants to estimate the coverage of a design principle by its assigned design best practices. In the following subsections, we discuss these three phases in more detail.

3.3.1. Brainstorming phase

In the brainstorming phase, a description of each design principle was provided. These descriptions were structured in the same way and included a definition, addressed design problem, technical details, affected quality aspects, and original definition of the principle. This information was presented by using so-called lessons using our Moodle system. Moreover, it was possible to download a design principle sheet containing the same information for each principle.

The goal of this phase was to build an understanding of the five design principles in each group. Therefore, the participants had to study the provided material and they were assigned to post their opinion, a question, or remark in an open group forum to foster interaction between them and critically reflect on the provided information. This is also the reason why this phase is called the brainstorming phase because brainstorming is a group creativity technique by which efforts are made to gather a list of ideas spontaneously contributed by the participants.

3.3.2. Clarification phase

Based on a solid understanding of the design principles, the clarification phase introduced a set of design best practices. These design best practices play an important role because they are used to measure design principles (Plösch et al., 2016a). However, there is no clear understanding of the importance of various design best practices and their positive or negative impact on principles. Consequently, this phase clarifies the relationship between design principles and the associated design best practices.

To address this uncertainty, the Delphi method was applied in two rounds. In the first round, a questionnaire containing the five design principles including their assigned design best practices was distributed. This questionnaire asked the participants to assess the importance of the design best practice in relation to the associated design principle. Accordingly, it was important to look at the design best practices from the perspective of the design principle. This concern was explicitly communicated to the participants. The assessment was conducted on a five-point scale from *very high (5)* to *very low (1)* including the *not relevant (0)* option. Additionally, the questionnaire asked the participants to list further design best practices that they consider to be relevant for measuring the design principle.

We collected the suggestions returned by the first round and removed duplicates. Further, it was necessary to align the suggestions to our naming conventions and description format. At this point, they were just suggested, but an assessment of the importance was still missing. Consequently, the new design best practices needed to go through the same assessment round as the proposed ones. With this additional step and the second questionnaire, the experts were asked (a) to verify that we had correctly interpreted their response and (b) to assess the importance of the suggestions. According to Schmidt (1997), "without this step, there is no basis to claim that a valid, consolidated list has been produced."

After this intermediate step, all design best practices were individually assessed by the participants. These responses were then consolidated into an interim result required for the second round. In more detail, the Delphi method specifies that the participants can revise their opinion based on the group opinion. This is why all participants received their questionnaire from the first round and the interim result. Then, they were asked to rework through the questionnaire and adjust their first importance assessment when they changed their perception. Finally, the revised and returned questionnaire represented the result of the clarification phase.

3.3.3. Completeness assessment phase

The third phase raised the complexity of the investigation to its top level since it aimed to derive the completeness of the design principles by their assigned design best practices. In more detail, a questionnaire asked the participants about two estimates for each design principle:

- (1) The coverage (in points) of a design principle by our proposed set of design best practices, and

[1] https://moodle.org/.

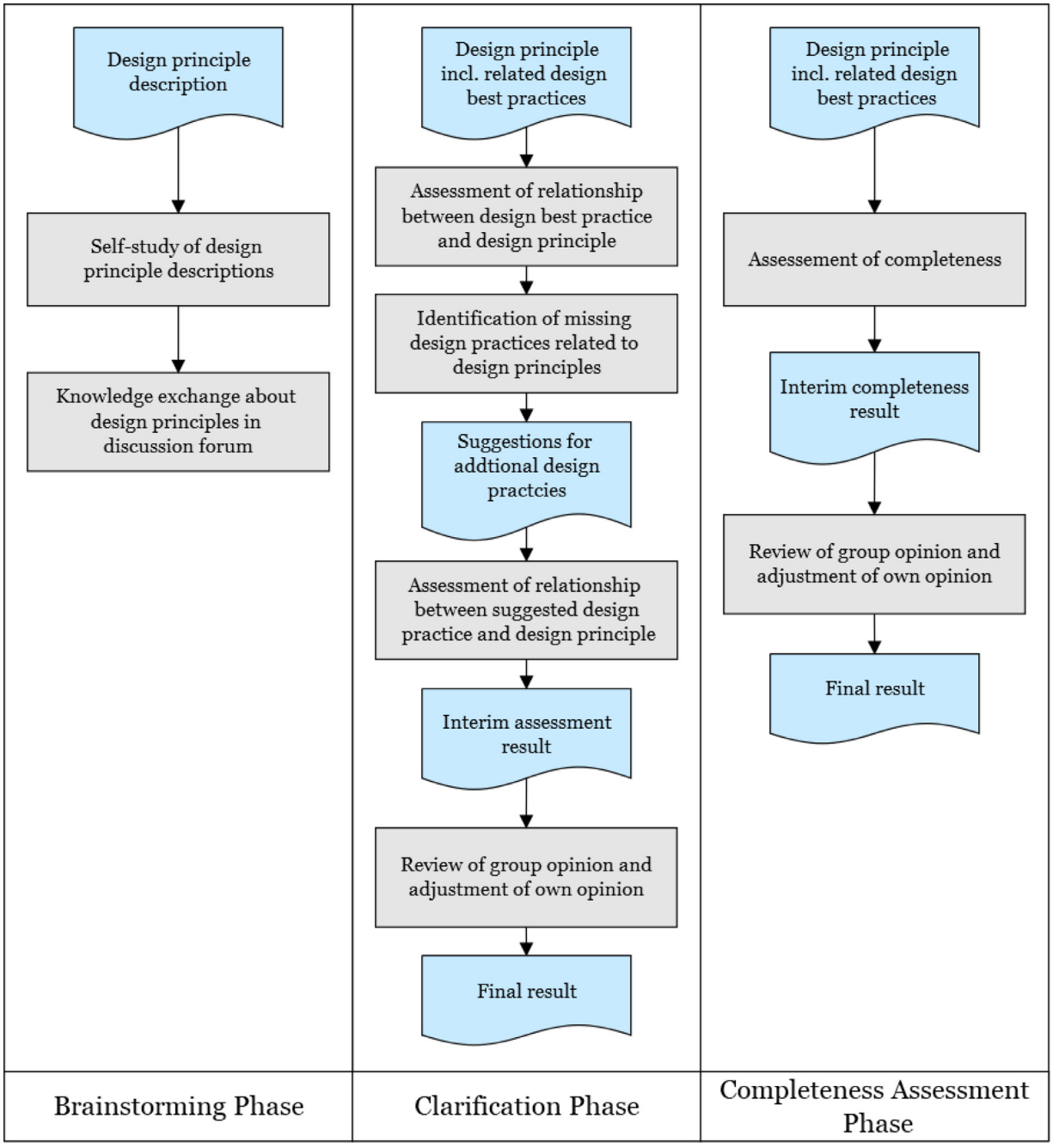


Fig. 1. Workflow of focus group discussion.

- (2) The coverage (in points) of a design principle by the design best practices suggested by the focus group.

For this task, the participants had to consider the derived importance of the design best practices as an indicator. To better explain this assignment, see the example in Fig. 2. This example assumes that a design principle is characterized by three design best practices from us and by two design best practices suggested by the study participants. Then, the participants had to assess to which degree the principle is covered by the proposed set and how much by the suggested set. Adding these two values does not necessarily need to sum to 100 points when there are still unmentioned aspects. Based on the example in Fig. 2, the participant assumes that DBP A, DBP B, and DBP C contribute 60/100 points to the principle, DBP Y and DBP Z (the two design best practices proposed by the study participants) get 30/100 points, and there are still unmentioned aspects, assessed by 10/100 points.

In the above example, the idea of unmentioned aspects needs some clarification. While designing this research and discussing its goal, it was identified that design principles are sometimes multifaceted, making it difficult for them to be fully covered by an enumerable set of design best practices. For instance, OCP states that software entities such as classes, modules, and functions should be open for extension, but closed for modification (Martin, 2003). Since this principle addresses most software entities, it is impossible to define all the characteristics that make up the principle. To address this circumstance and avoid a too rigid assessment of completeness, we provided this evaluation buffer, which is more or less based on the gut feeling of the participants.

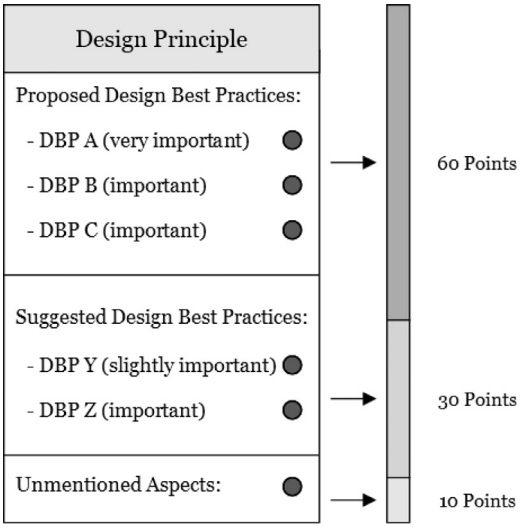


Fig. 2. Coverage of design principles by their assigned design best practices.

Following the Delphi approach, the assessment in this phase was again conducted in two rounds where the participants received the responses of the other participants to revise their assessment. Finally, the adjusted questionnaires represented the result of the focus group.

4. Results

In the brainstorming phase, the participants studied the provided material on the design principles and contributed to a forum discussion. This discussion aimed to build an understanding but without any intent to be further analyzed. The main data collection happened in the second and third phases.

4.1. Suggestions for additional design best practices

While the assessment of the importance of design best practices on design principles in the second phase relied on a defined scale, the open question asking about additional design aspects was analyzed qualitatively. Thus, the notes from the participants were examined by two independent researchers to derive their design purpose and to identify similarities to other practices. Without judging the objective of the suggestion, we transformed the notes into a design rule description and a meaningful name. Therefore, we aligned the suggested description to our pattern of delineating design best practices and exchanged terms as used in other descriptions. Further, the original note, derived description, and name were verified by the proposer to reveal misinterpretations. In total, 34 additional design best practices were suggested from which two conceptually overlapped with the practices from our pool.

It is important to understand that the focus groups were working separately and that new ideas were not shared among the other teams. Consequently, each discussion contained just a subset of the 34 suggestions, as depicted in Tables 1 and 2. The descriptions for each suggested design best practices are listed in the Appendix B. Furthermore, Table 2 shows that none of the members in FG-V provided a suggestion for an additional design best practice. Nevertheless, these two tables summarize the answer to RQ2, which aims at identifying missing design best practices.

For these new ideas of design best practices, we did not require that they can be automated with a static code-measuring tool. The design best practice *UseMeaningfulVariableNames*, for instance, is related to the naming conventions of variables and requires that a variable must have a meaningful name. A static code analyzer could verify the length and notation of the variable name, but meaningfulness is hard to grasp without knowing the underlying semantics. Thus, static code analysis is insufficient for implementing all the suggestions as an automated rule.

Table 1
Suggested design practices for IHI, SRP, FCOI, CCP, and CQS.

DP	Focus Group	Design Best Practice
IHI	FG-I	UseImmutableObjects DontExposeInternalStructureOfClass AvoidExposingImplementationDetailsInMethodN.
	FG-III	MakeClassesPublicIfNecessary CheckInterrelatednessOfLayer
	FG-V	AvoidReturningCollectionsArrays
SRP	FG-I	AvoidDuplicationOfStateInformation CheckMethodUsageDynamically
	FG-III	AvoidUnbalancedInheritanceAndDelegationHier. AvoidUnrelatedFields
	FG-V	AvoidStaticMethods AvoidNonCohesiveInterfaces
CCP	FG-I	UseCoherentNaming
	FG-III	AvoidSharedClassesInSubPackages AvoidSimplyDependenciesAcrossMultiplePackages
CQS	FG-I	AvoidReturningContainerObjectsFromCommands
	FG-III	AvoidMutableFieldsWhenPossible AvoidPretendedObjectStates
FCOI	FG-I	AvoidLargeObjects

Table 2
Suggested design practices for DRY, ADP, OCP, and ISP.

DP	Focus Group	Design Best Practice
DRY	FG-II	UseMeaningfulVariableNames CheckPrivateMethodUsage CheckGroupingOfUtilityClasses
	FG-VI	AvoidSameInformationInDifferentArtifacts AvoidDeadCode
ADP	FG-II	AvoidNonCohesivePackageImplementations UseStrictLayering KeepInterfaceImplementationsTogether
	FG-VI	AvoidReferencingImplementationPackages AvoidPropertyInjection
ISP	FG-II	CheckUnrelatedMethods
	FG-VI	AvoidInterfaceInheritance
OCP	FG-II	UseInterfaceForExternalPackageDependencies

4.2. Relative importance of the design best practices

At the end of phase two, the relative importance of all design best practices to their assigned design principle had been derived. For this assessment, the participants had to provide the first opinion on a five-point scale and could then revise their opinion in a second round. This provided the opportunity to consider other opinions and critically reflect on the first opinion if it significantly differed from the group understanding. While the Delphi method defines this reflection round, the participants changed their first assessment in only a few cases.

The result of this phase is a list of 15 opinions about the relative importance of each practice to a principle. We aggregated all individual judgments to one representative group opinion by first transferring the ordinal scale values to their numerical representation and then calculating the arithmetic mean. This mean was then mapped onto the five-point scale - *very high (5), high (4), moderate (3), low (2), and very low (1)* - since it is sufficient

Table 3
Design best practices for IHI, SRP, FCOI, CCP, and CQS.

DP	Design best practice	Importance
IHI	AvoidPublicFields	very high
	DontReturnMutableCollectionsOrArrays	very high
	AvoidUncheckedParametersOfSetters	high
	UseInterfacesAsReturnType	high
	AvoidProtectedFields	moderate
	AvoidSettersForHeavilyUsedFields	moderate
	AvoidManySetters	low
	AvoidManyGetters	very low
SRP	AvoidNonCohesiveImplementations	very high
	CheckUnsuitableFunctionalityOfClasses	high
FCOI	CheckUnusedSupertypes	very high
	UseCompositionNotInheritance	very high
CCP	AvoidNonCohesivePackages	very high
	AvoidStronglyCoupledPackages	very high
	AbstractPkg.ShouldNotDependOnOtherPkg.	high
CQS	AvoidCommandsInQueryMethods	very high
	DontReturnUninvolvedDataFromCommands	very high
	AvoidReturningDataFromCommand	low

Table 4
Design best practices for DRY, ADP, OCP, ISP, and PINI.

DP	Design best practice	Importance
DRY	AvoidDuplicates	very high
	DocumentInterfaces	very high
	DocumentPublicClasses	high
	AvoidSimilarAbstractions	moderate
	DocumentPublicMethods	moderate
	AvoidSimilarNamesForSameDesignElements	moderate
	AvoidSimilarNamesForDifferentDesignElemen.	moderate
	AvoidMassiveCommentsInCode	low
ADP	AvoidPackageCycles	very high
OCP	AvoidUncheckedParametersOfSetters	moderate
	AvoidPublicStaticFields	moderate
	DontReturnMutableCollectionsOrArrays	moderate
	UseInterfacesAsReturnType	moderate
	AvoidPublicFields	moderate
	AvoidRuntimeTypeIdentification	moderate
	UseAbstractions	low
	AvoidSettersForHeavilyUsedFields	very low
	AvoidProtectedFields	very low
ISP	CheckUnsuitableFunctionalityOfClasses	high
PINI	UseInterfacesIfPossible	high
	ProvideInterfaceForClass	high

and easier to communicate this value representation. We also offered the answering option *not relevant (0)* that was not used by any of the participants of the focus groups. For the mapping we defined the range of each ordinal scale value in between $+/-0.5$ points around the default value; for instance, the arithmetic mean of 3.3 was mapped to *moderate (3)*. We did not calculate the arithmetic mean of the suggested design best practices since they were not distributed among all groups and just discussed within each group. Tables 3 and 4 show the design best practices and their group opinion for all 10 design principles and summarize the findings to answer RQ1.

Except for OCP, the research reveals that the other nine principles have at least one design best practice, which has high or very high importance. In other words, those design best practices assessed as very high or high meet the design aspect of the principle. This provides a first justification that practices express certain parts of the associated design principles. However, to understand the power of the proposed and suggested practices, we asked the participants about their estimation.

Table 5
Completeness assessment of IHI, SRP, FCOI, CCP, and CQS.

DP		FG-I		FG-III		FG-V	
IHI	(1)	70.0	10.0	56.0	16.2	74.0	7.4
	(2)	26.3	9.6	11.0	2.0	12.0	4.0
	(3)	3.7	4.1	33.0	15.4	14.0	3.7
SRP	(1)	65.0	16.6	56.0	23.3	54.0	4.9
	(2)	17.5	8.3	15.0	4.5	32.0	4.0
	(3)	17.5	14.8	29.0	22.0	14.0	4.9
FCOI	(1)	87.5	10.3	73.0	16.0	88.0	4.0
	(2)	6.5	8.0	–	–	–	–
	(3)	6.0	10.4	27.0	16.0	12.0	4.0
CCP	(1)	82.5	4.3	52.0	16.0	82.0	4.0
	(2)	8.0	4.9	18.0	5.1	–	–
	(3)	9.5	7.3	30.0	20.2	18.0	4.0
CQS	(1)	85.0	5.0	54.0	12.0	92.0	9.8
	(2)	12.5	2.5	30.0	8.9	–	–
	(3)	2.5	2.5	16.0	13.6	8.0	9.8

Table 6
Completeness assessment of DRY, ADP, OCP, ISP, and PINI.

DP		FG-II		FG-IV		FG-VI	
DRY	(1)	64.0	4.9	77.6	2.2	50.4	14.8
	(2)	26.0	3.7	–	–	32.6	11.5
	(3)	10.0	7.1	22.4	2.2	17.0	12.4
ADP	(1)	66.0	15.9	78.6	8.7	54.0	6.9
	(2)	34.0	15.9	–	–	24.0	9.6
	(3)	0.0	0.0	21.4	8.7	22.0	9.4
OCP	(1)	73.0	4.0	80.6	16.4	68.0	14.6
	(2)	18.0	6.8	–	–	–	–
	(3)	9.0	6.6	19.4	16.4	32.0	14.6
ISP	(1)	45.0	4.5	80.8	11.0	44.0	14.9
	(2)	46.0	10.2	–	–	29.0	16.3
	(3)	9.0	9.2	19.2	11.0	27.0	11.7
PINI	(1)	89.0	12.0	80.8	10.3	72.0	25.7
	(2)	–	–	–	–	–	–
	(3)	11.0	12.0	19.2	10.3	28.0	25.7

4.3. Completeness achieved by the design best practices

To understand the completeness of the design principles by their assigned design best practices, the participants had to divide a cardinal scale, which ranged from 0 to 100, into three parts, representing (1) the completeness obtained by our proposed set of design best practices, (2) the completeness obtained by the suggested set of design best practices, and (3) the unmentioned aspects of the design principle not addressed by any of the practices. Section 3.3, as well as Fig. 2, explain this task by means of an example.

By applying the same approach as in phase two, the participants had to provide the first opinion on completeness. Since we were aware of the difficulty of this assignment, we motivated the task as relying on the gut feeling for this first estimation. After the group members returned the assignment sheet, we summarized the opinions into one interim group result. This result was individually distributed to the participants, who had to reflect on the group opinion critically. If the view of a participant changed during this reflection process, he or she was allowed to alter the initial assessment by changing the distribution of the three parts.

Lastly, we summarized all the revised assessments into one final group opinion. Tables 5 and 6 depict the final group opinion of all six focus groups separated by the two sets of design principles. The first row (1) in each design principle shows the completeness achieved by our proposed set of design best practices, the second

row (2) the additional completeness achieved by the suggested set of design best practices, and the third row (3) represents the unmentioned aspects of the design principle. If a focus group did not provide any suggestion for a particular principle, the assessment of this part could not be conducted, resulting in an empty cell in the second row.

According to Table 5, the design principle CQS achieves the highest completeness assessment with 92.0 points by FG-V. In contrast, FG-III judges the design principle CCP with the lowest completeness value of 52.0 points in relation to our proposed set of design best practices. When considering the coverage achieved by the suggested practices of FG-III for CCP (18.0), this particular design principle reaches 70.0 points. In Table 6, the highest completeness assessment achieves the design principle PINI with 89.0 points by FG-II. Contrarily, the design principle ISP has only 45.0 points by the same focus group. The reason for this relatively low number may be the high assessment of suggested design best practices with 46.0 points.

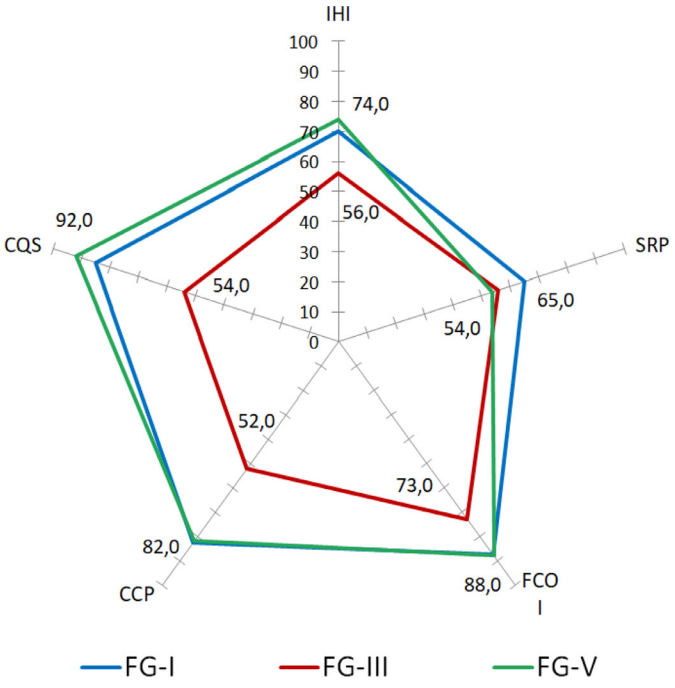
To better compare the completeness assessments, Figs. 3(a) and (b) present these data by using a net chart. Each chart has five dimensions, one for each particular design principle. Further, the dimensions range from 0 to 100 points, representing the cardinal scale that needed to be divided into three parts. For this analysis, we concentrate on the completeness achieved by our proposed set of design best practices. Thus, we marked the assessments of the three focus groups for a particular design principle on its related dimension. For instance, the design best practices for IHI were assessed with 56, 70, and 74 points by the individual focus groups as shown in the vertical dimension in Fig. 3(a). The three shapes in the chart were drawn by connecting the values for each focus group.

As shown in Fig. 3(a) by the green and blue shapes, two focus groups almost perfectly agree in their assessment of the completeness of the five design principles. Both assess the completeness of CQS and FCOI as very high with a value of around 92 and 88 points, respectively. CCP and IHI achieve good completeness (around 82 and 74 points), while SRP achieves a completeness of around 55 points. The third focus group, which is represented by the red shape, is more critical regarding completeness. Thus, four principles achieved a completeness of around 54 points and only for FCOI does their perception go up to 73 points.

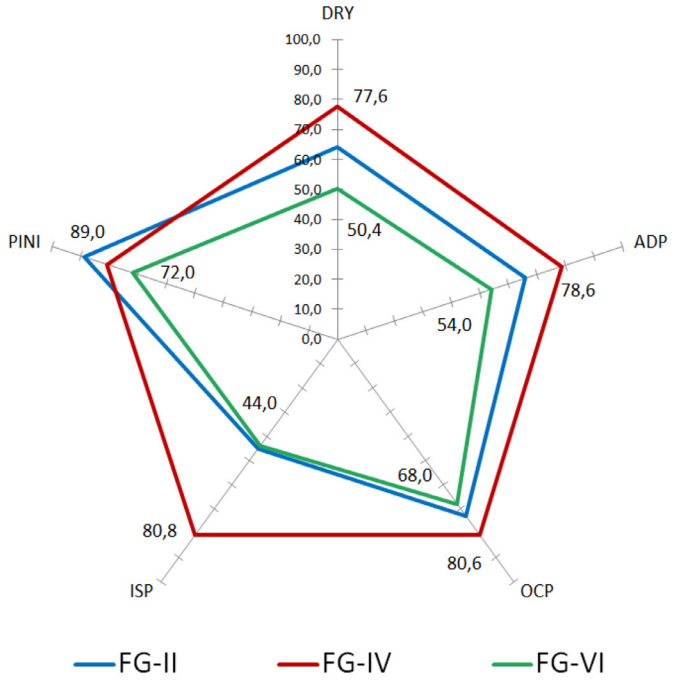
Similar to the result of the first set of design principles, two focus groups correlate in their opinion about the completeness of the other five principles. Fig. 3(b) displays these two groups by the green and blue shapes. It is visible in Fig. 3(b) that the group in green is more critical compared with the other group, but both focus groups assign high completeness to PINI and OCP. The principles DRY and ADP receive moderate completeness (around 50 points), while ISP has the lowest completeness with 44 points.

The focus group displayed by the red line assesses the completeness of all five principles to be around 80 points. This group was also modest in providing suggestions for additional design best practices. Due to this reticence and without considering missing aspects, we assume that they tend to assess our proposed set of design best practices with a higher contribution to completeness. This effect is evident concerning ISP. For this principle, the other groups provided suggestions and assessed these suggestions with a contribution to completeness around 40 points, but FG-IV did not consider these missing aspects. The consequences of this observation are discussed below.

Figs. 4(a) and (b) provide more details about the coverage of a particular design principle including the set of suggested best practices. More specifically, the dashed line of each focus group represents the assessment of coverage achieved by our set of practices (same information as depicted in Figs. 3(a) and (b)) and the corresponding solid line considers the set of new practices. According



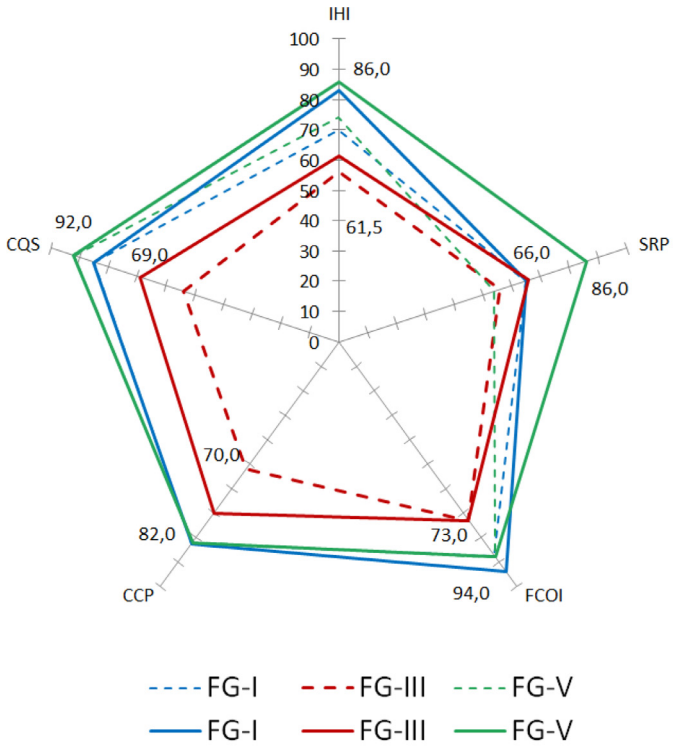
(a) Comparison principle set I



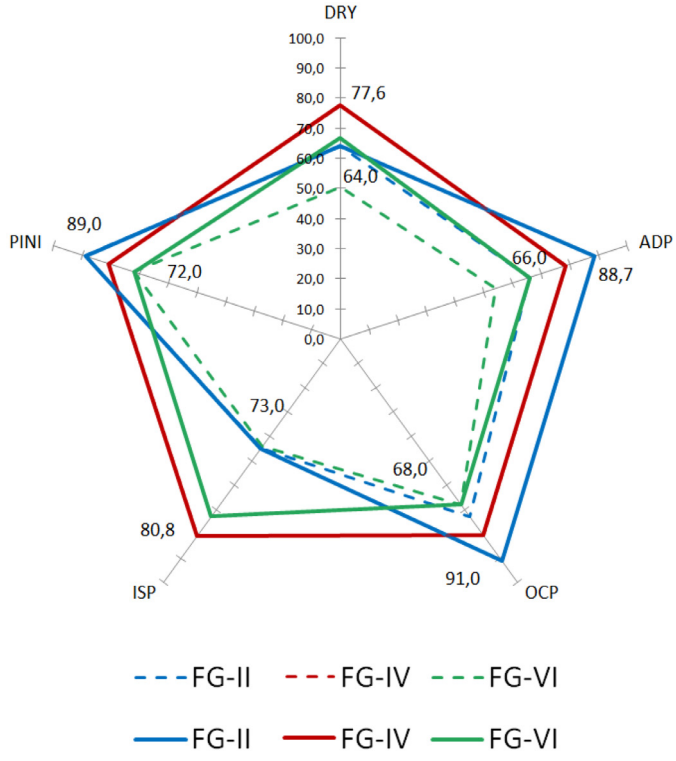
(b) Comparison principle set II

Fig. 3. Comparison of design principle coverage. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

to Fig. 4(a), IHI and SRP can gain most increment with a final coverage of 86 points based on the judgment of FG-V. Generally, FG-III provided many new ideas for design best practices resulting in at least 61.5 points for all five principles. A similar picture is shown for FG-VI in Fig. 4(b) with an increase of coverage for all five principles to at least 64 points. The principles that achieve the highest coverage in Fig. 4(b) are PINI, ADP, and OCP with an assessment around 90 points based on the opinion of FG-II. In Fig. 4(b), it looks



(a) Consideration of suggestions for principle set I



(b) Consideration of suggestions for principle set II

Fig. 4. Consideration of suggestions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 7
Average completeness of the design best practices.

DP	Mean	Std. Dev. in Points	DP	Mean	Std. Dev. in Points
IHI	66	14	DRY	65	14
SRP	58	17	ADP	74	16
FCOI	83	13	OCP	65	14
CCP	71	18	ISP	57	20
CQS	76	19	PINI	81	19

like the red dashed line is missing but in fact it is hidden by the red solid line. This fact results from the circumstance that FG-IV did not suggest additional best practices as mentioned above.

In order to answer RQ3 that focus on deriving a representative group opinion about the completeness of each design principle, we calculated the average of the assessments including the standard deviation. Table 7 presents this result. According to this table, FCOI has the highest completeness value and lowest standard deviation. In other words, the groups agree that the design best practices address the main design concerns of this principle. On the contrary, ISP has the lowest completeness value and highest standard deviation. Thus, there is still some uncertainty whether the proposed design best practice (*CheckUnsuitableFunctionalityOfClasses*) is sufficient to grasp the intent of the principle.

5. Discussion and lessons learned

This part discusses the applied research method and final results. Therefore, this section is subdivided into the three phases defined by the discussion process. Before diving into the three phases, we provide some general remarks on the focus group research approach combined with the Delphi method.

- *Plan enough time resources:* Conducting this online focus group research was time-consuming. First, the groups started time-delayed. Although this aspect distributed the workload for handling the groups, the overall duration increased. In more detail, we began the first group on 7th February and received the last reply on 28th April 2017. Second, the participants responded slowly. We originally planned a focus group discussion for four weeks, with the second and third phases to be completed within ten working days. Finally, the average duration of each group amounted to seven weeks.
- *Manage discussion with tool support:* In total, 31 experts participated in this research. They were divided into six groups, which started on different dates and ran simultaneously. Managing these groups without proper tool support would have been difficult because they also had another response behavior, resulting in various progress levels. To keep the discussions on track, we used the course and content management tool Moodle. In combination with email for personal notifications, Moodle was appropriate for managing the groups. At this point, it is important to note that personalized communication should be preferred when possible. The reason is that we observed faster-responding behavior when assignments were individually issued.
- *Consider potential dropouts:* In Kontio et al. (2008), the authors argue that focus group participants are more likely to stay with the discussion process since they provided an initial commitment to contribute. In our research, however, we observed that two members disconnected from the group discussion and could no longer be reached. Fortunately, that happened in the early phases of the first focus groups, so that the places were filled with participants waiting for their discussion to start. In other words, participants from discussions that

had not yet started replaced these dropouts. Before replacing a disconnected group member, a background check ensured that the new participants matched the characteristics of the others to keep the homogeneity of the group. As a consequence, it is important to have a solid strategy to handle dropouts without threatening the research design.

5.1. Brainstorming phase

The entire discussion started by introducing the topic of design principles and providing self-study material. We encouraged the participants to work on the provided material and participate in the group forum. The discussions in these forums were interesting even though some participants were reserved in expressing their opinions. Nevertheless, all participants fulfilled the first task, ensuring a good common level of understanding.

5.2. Clarification phase

The result of the first task in the second phase shows that a few groups were restrained in providing a suggestion for new design best practices. The most obvious group in this regard was FG-IV, with no single suggestion at all. Forcing the groups to provide at least one idea for each principle would have distorted the research, so we did not intervene in this regard. Further, the research method does not provide a recommendation for this issue. This is why we continued the discussions with the individual group contributions.

To assess the relative importance of each practice, the suggestions do not have any impact since each practice was individually assessed. In other words, the evaluation task in this round did not ask the participants to rank the proposed and suggested design best practices, but rather to express the relative importance of each practice separately. Consequently, it is possible to determine a representative group importance for our proposed design best practice because the participants provided their assessment in the same circumstances.

Before this focus group research, we conducted a survey on the general importance of design best practices, but it did not consider the principle relation (Bräuer et al., 2017b). The result derived from this survey returned a ranking of the design best practices and an own importance range based on the standard deviation. This importance range can be used for up- or downgrading a practice depending on the project context, which differs according to project requirements, used frameworks, and the criticality of design quality. To further analyze the result of this focus group research, we mapped the survey result with the result of this investigation. Therefore, we checked each practice to assess whether its relative importance is within its general range. This was shown for all except four proposed design best practices.

The first practice is *DontReturnMutableCollectionsOrArrays* that has an importance range from low to high; however, it is considered to be very high in the context of IHI. The same applies for *CheckUnusedSupertypes* in regard to FCOI. The other two exceptions are assessed in the context of OCP. As briefly mentioned above, the investigation did not reveal a design best practice with at least a high importance for OCP. Consequently, the two design practices *AvoidUncheckedParametersOfSetters* and *AvoidPublicFields*, which have an importance range from high to very high, received a downgrade to a moderate assessment in the context of OCP.

Comparing the survey with the focus group result at the principle level presents a minor difference for IHI and DRY. Table 8 shows this deviation. In the context of IHI, this means that practices – except *AvoidManyGetters* – received an equal or higher relative importance compared with the survey result. For instance, *DontReturnMutableCollectionsOrArrays* is considered to be very important even though the survey concluded a moderate importance. For DRY, this picture is slightly different because the focus group members tended to be more distinctive. Thus, they assessed *AvoidDuplicates, DocumentInterfaces*, and *DocumentPublicClasses* with a similar importance level as derived from the survey, but the other practices were downgraded to medium or low.

Table 8
Deviation at the principle level.

DP	Design Best Practice	FG Result	Survey Result
IHI	AvoidPublicFields	vh	vh
	DontReturnMutableCollectionsOrArrays	vh	m
	AvoidUncheckedParametersOfSetters	h	h
	UseInterfacesAsReturnType	h	h
	AvoidProtectedFields	m	l
	AvoidSettersForHeavilyUsedFields	m	l
	AvoidManySetters	l	l
	AvoidManyGetters	vl	l
DRY	AvoidDuplicates	vh	vh
	DocumentInterfaces	vh	h
	DocumentPublicClasses	h	h
	AvoidSimilarAbstractions	m	h
	DocumentPublicMethods	m	h
	AvoidSimilarNamesForSameDesignEl.	m	h
	AvoidSimilarNamesForDifferentDesignE.	m	h
	AvoidMassiveCommentsInCode	l	m

very high (vh), high (h), moderate (m), low (l), very low (vl).

Summarizing, in almost all cases the same or a higher importance level was given by the focus group participants. We consider this to be quite natural, as the relation of a design best practice to a more specific design principle is easier to grasp than the relation of the design best practice to general design quality. The design best practice *AvoidMassiveCommentsInCode* is one exception to this rule. The basic underlying idea of DRY in this context is to choose an appropriate code structure with appropriate naming and a good documentation of the application programmer interface and not to comment the implementation of methods. This practice seems to be counterintuitive for the focus group participants.

5.3. Completeness assessment phase

The result of the completeness evaluations of the focus group provides some space for discussion, especially the effect that two groups in each principle set came to almost the same assessment for our proposed set of practices.

5.3.1. Discussion of outliers in the first principle set

In the first set of design principles, FG-I and FG-V inferred nearly the same assessment of completeness. One variable that could have affected the completeness assessment is the number of suggestions the groups provided for each principle. Tables 1 and 2 depict this information. As shown there, FG-I suggested additional practices for all principles, while FG-V provided suggestions just for IHI and SRP. Thus, it is interesting whether FG-V considered the missing aspects even though they did not submit specific practices. In fact, the missing aspects were taken into account, as represented by the assessment of the uncovered design aspects. Good examples of this observation are FCOI and CCP. Overall, we concluded that the number of suggestions did not affect the perceptions of the two groups.

Based on this finding, it can be assumed that FG-I and FG-V are similar in their characteristics but differ from FG-III in another way. Consequently, the standard deviations within each group were analyzed. This analysis shows that FG-III has the highest standard deviation for each principle. In other words, FG-III is the most heterogeneous group in this principle set. To be more specific, the lowest

Table 9
Contribution of suggestions I.

FG	Design Best Practice	relative Importance	Contribution to Completeness
FG-I (26.3)	UseInterfacesIfPossible	4.2	6.6
	UseImmutableObjects	2.4	6.6
	DontExposeInternalStructureOfClass	4.0	–
	AvoidExposingImpl.DetailsInMethodN.	4.0	–
FG-III (11.0)	MakeClassesPublicIfNecessary	4.6	5.5
	CheckInterrelatednessOfLayer	3.2	–
FG-V (12.0)	AvoidReturningCollectionsArrays	3.6	12.0
FG-I (17.5)	AvoidDuplicationOfStateInformation	3.6	–
	CheckMethodUsageDynamically	2.0	–
FG-III (15.0)	AvoidUnbalancedInheritance.Hierarchie	3.6	–
	AvoidUnrelatedFields	4.4	5.0
	UseCompositionNotInheritance	5.0	5.0
FG-V (32.0)	AvoidStaticMethods	4.4	16.0
	AvoidNonCohesiveInterfaces	3.4	16.0
FG-I (6.5)	AvoidLargeObjects	2.8	6.5
FG-I (8.0)	UseCoherentNaming	1.4	–
FG-III (18.0)	AvoidSharedClassesInSubPackages	4.0	9.0
	AvoidSimplyDep.AcrossMultiplePkg.	2.8	9.0
FG-I (12.5)	AvoidReturningCont.Obj.FromComm.	1.4	–
FG-III (30.0)	AvoidMutableFieldsWhenPossible	4.6	15.0
	AvoidPretendedObjectStates	5.0	–

standard deviation for FG-III is 12.0 points, while the standard deviation for none of the principle in FG-V goes beyond 10 points. Except for SRP that has a standard deviation of 16.6 points (the average assessment of FG-III for SRP deviates by 23.3 points), the same applies to FG-I. Owing to this finding, it is evident that two or three members in FG-III tended to have a more critical view of the assessment of completeness. As a conclusion, the general opinion of FG-III would move closer to the others when lowering the critical perceptions.

5.3.2. Discussion of outliers in the second principle set

In the second set of design principles, FG-II and FG-VI correlate in their assessment, while FG-IV represents the outlier. As briefly mentioned above, the fact that none of the participants in FG-IV provided any suggestion for a new design best practice supports the idea that deep involvement in this topic did not happen. Consequently, a representative group opinion of completeness for DRY, ADP, OCP, ISP, and PINI is probably too optimistic and should be downgraded when used in practice.

5.3.3. Consideration of suggestions

The best practices suggested in the second phase are a valuable input to enhance our measurement tool MUSE. To evaluate whether they can be implemented and checked automatically, three MUSE developers individually examined the underlying idea of each suggestion. If one of the developers derived a different opinion about the implementability, a joint discussion concluded a shared understanding of the suggestion and a final agreement on the potential to be automated.

Next to the requirement of being able to be implemented in MUSE, the contribution of suggestions to the completeness of the principle was derived. To do so, we divided the group assessment by the number of suggestions for each principle and each group. For instance, FG-I concluded that its four suggestions for IHI amount to 26.3 points. In other words, one automated suggestion provides a contribution to completeness with 6.6 points, when dividing this assessment by the number of suggested best practices (26.3 divided by 4 suggestions). Tables 9 and 10 depicts the suggestions that can be implemented by the last column. Hence, the cell is left empty for suggestions that cannot be automated; otherwise, the cell represents the contribution to completeness.

Given the information on the contribution to the completeness of a principle by each rule, we can derive candidates that are worth being implemented. For instance, *AvoidInterfaceInheritance* represents an important design aspect for ISP according to FG-VI. Other examples are *AvoidStaticMethods* and *AvoidNonCohesiveInterfaces* with a high contribution to express SRP or *UseInterfaceForExternalPackageDependencies* as another design aspect of OCP.

While Tables 9 and 10 depict the relative importance of the implementable suggestions, they also show the relative importance of non-automatable rules. By using this information, a quality manager can derive alternative ways of checking these practices, especially those with high relative importance. The suggestion *AvoidPretendedObjectStates*, for example, is considered to be very important for CQS but it cannot be implemented in MUSE. Subsequently, it makes sense to think about other techniques to check this rule in the source code. Additional candidates are *UseMeaningfulVariableNames* and *CheckUnrelatedMethods* that both require understanding the underlying semantics of the variables and methods, respectively.

5.3.4. Uncertainty for interpreting the design principles

During the focus group research, some participants expressed uncertainty about the interpretation of the principles. To understand this issue, we examined the average of the assessments reflecting the uncovered aspects of each design principle. Ordering the principles according to this value returned a ranking, as depicted in Table 11. According to the table, OCP, SRP, and CCP have the highest degree of uncovered design aspects. This could infer that their definitions still leave space for interpretation and are difficult to grasp. Otherwise, the groups would have provided a suggestion for design best practices or would have assessed the uncovered aspects as low.

On the opposite side of the ranking are CQS, ADP, and FCOI with the lowest assessments of uncovered design aspects. Either they have been addressed with suggestions or the principle is pre-

Table 10
Contribution of suggestions II.

FG	Design Best Practice	relative Importance	Contribution to Completeness
FG-II (26.0)	UseMeaningfulVariableNames	4.8	–
	CheckPrivateMethodUsage	2.2	–
	CheckGroupingOfUtilityClasses	3.4	–
FG-VI (32.6)	AvoidSameInformationInDiff.Artifacts	4.0	–
	AvoidDeadCode	4.2	16.3
FG-II (34.0)	AvoidNonCohesivePkg.Implementations	3.4	11.3
	UseStrictLayering	3.8	–
	KeepInterfaceImplementationsTogether	3.2	11.3
FG-VI (24.0)	AvoidReferencingImplementationPkg.	3.7	12.0
	AvoidPropertyInjection	2.2	–
FG-II (18.0)	UseInterfaceForExternalPackageDep.	4.4	18.0
FG-II (46.0)	CheckUnrelatedMethods	4.8	–
FG-VI (29.0)	AvoidInterfaceInheritance	3.5	29.0

Table 11
Principle ranking.

DP	Mean of uncovered aspects
OCP	20.8
SRP	20.4
CCP	19.9
PINI	18.8
ISP	18.5
IHI	17.9
DRY	16.1
FCOI	15.6
ADP	15.4
CQS	9.3

cisely enough defined so that the appropriate design best practices were identified beforehand. Regardless of which fact, we conclude that these principles are clearer and can be followed by design best practices.

6. Limitations

In any experimental study, some factors influence the findings and represent threats to validity. In more detail, threats to internal validity concern any confounding variables that could have influenced the outcome (Wohlin et al., 2012). The decision to conduct online focus group discussions instead of on-site (face-to-face) discussions represents such a threat in our perception. Thus, communication via text is less rich because it misses body language or facial expression and text can be misunderstood (Kontio et al., 2008). For the sake of anonymity, we accepted the lack of body language in discussions. Further, the open discussion part was reduced to the brainstorming phase, which focused on getting participants to the same level. To address the problem of misunderstandings, we explicitly consulted participants to clarify, for example, a suggested design best practice.

Another internal threat to validity is the selection and segmentation of participants. To control this issue, we checked the software engineering skills of the volunteers by using a previous survey. More specifically, participants who assessed their object-oriented programming expertise as good or top were invited to participate in a focus group. The re-participation of these participants may result in the issue that they have a knowledge advance compared to the others. To mitigate this threat, we provided the same information to the participants, who got in touch with our design best practices for the first time. The three focus groups arranged by our research partner contained senior software engineers with top engineering competencies and with ambitions to consolidate their software design knowledge. Moreover, the segmentation of participants was conducted in a way that focused on ensuring homogeneity among the teams. Therefore, members of the same company were part of the same group.

The most critical internal threats may have been introduced by the provided material and the proposed relationships between the design principles and design practices. Each design principle was explained by using a separate lesson in Moodle, which systematically structured the information. However, we did not verify the understanding of the design principles but rather asked the participants to discuss an aspect or example of each principle in a group forum. Based on these discussions, we could ensure that everybody understood the main intention of each principle. Confronting the participants with a mapping between principles and practices suggests a relevant relationship to some extent. Consequently, a participant would tend to not argue against this relationship. However, we observed examples where a group voted against a proposed relationship between a principle and practice, thus showing their critical reflection of the assignment.

Threats to external validity concern the ability to generalize the results (Wohlin et al., 2012). In this regard, we do not see a major threat because the design principles were discussed independently of object-oriented programming languages and detached from any application context. Nevertheless, we currently know a set of 67 design best practices (see Plösch et al. (2016b)) containing the rules for Java, C++, and C# (some of these rules are used to measure design principles not mentioned in this article). Consequently, these findings can be transferred to other programming languages but with particular care; for C++, additional design features such as multiple inheritance or macros are available.

The number of participants may also represent a limitation in this study. In total, two sets of five principles were separately discussed by 15 (+1) participants. While it can be argued that 15 opinions to one principle are still too few, we observed that the complexity of the topic presupposes an in-depth examination of the design principles that cannot be achieved by conducting a broad investigation in the form of, for example, an online survey. This is not possible because critically reflecting on design principles and design best practices cannot be performed within the short time given by a typical survey setting. Consequently, we conclude that a focused discussion of a small group on one topic reveals more valuable insights.

7. Conclusion and future work

Design principles in software engineering are essential for software developers and designers because they communicate design

knowledge for building software that ensures internal quality attributes (-ilities) such as maintainability, functional suitability, and portability. However, they are too vague to be appropriately applied in practice. Therefore, we aim to operationalize design principles by using design best practices, which in turn are concrete enough to be followed. Although we have previously built a design quality model that reflects the relationship between design principles and the assigned best practices (Plösch et al., 2016a), we are not aware of the strength of the relationship or whether we forgot design best practices for a particular principle. Due to these remaining research questions, this investigation was undertaken and returned following key contributions for the community.

First, the focus group research derived a clear picture of the importance of design best practices in relation to the assigned design principles. This picture rests upon the opinions of at least 15 senior software engineers (five groups of five and one of six participants – each assessing five design principles) with a solid awareness of software design. Based on the findings, design improvement actions can be preselected and prioritized to invest in the effort that has the highest improvement impact. Assuming a project team wants to enhance its compliance with the IHI, the avoidance of public fields is more important than protected fields. Thus, this gained information is used to guide improvements to be most effective.

The findings have relevance not only for design improvements but also for the quality task of design assessments. For instance, and by using the above example, public fields in a class are more critical compared with protected fields. In fact, the weight is determined by the derived importance level and can be used to assess, for example, the compliance of information hiding. In Plösch et al. (2016a), we show an instantiation of a design quality model suitable for this purpose and that can be adjusted based on the derived weights.

Another contribution is the identification of missing design practices that affect a particular design principle. In fact, 32 additional design practices were suggested by the six focus groups. While some of the suggested design practices represent manual measures, meaning that they cannot be automated, a set of 18 practices could be implemented in a design analysis framework, particularly in our MUSE tooling. The agenda of our future work contains the implementation of these suggestions, including a joint assessment of their relevance for a design principle. Hints for this judgment can be derived from the group opinion of the team that suggested the practice.

Finally, the assessment of the obtained completeness of each design principle shows that most design aspects are covered by design best practices. In other words, none of the principles is missing a central element. Moreover, the high degree of rule automation depicts that the operationalization of design principles can be conducted automatically.

For future research avenues, we plan to examine the design quality model based on the gained findings from this study within an industrial setting. By using the technical action research approach proposed by Wieringa and Moralı (2012), the artifact – the design quality model – will be evaluated regarding its suitability and usefulness for design quality assessments. Furthermore, we have published a (benchmark-based) design improvement portfolio approach that addresses the quality task of guiding improvement actions (Bräuer et al., 2017a). This approach has been introduced at the level of design best practices. Now, the results from this study can be incorporated into the portfolio technique to provide better recommendations at the level of design principles. An evaluation of this design improvement guidance is pending. In general, a better understanding of design principles is essential for the software engineering discipline since they are strong design guidelines as indicated by Stevenson and Wood (2017). Hence, future work has to focus on this aspect.

Appendix A. Design best practices

- *AbstractPackagesShouldNotDependOnOtherPackages:* A package containing a high number of abstract classes and interfaces should not depend on other packages.
- *AvoidCommandsInQueryMethods:* A public method identified as query method should not change the state of the object and can only call query methods of the same class. A public method is identified as query method when its name starts with a defined prefix such as get, has, or is.
- *AvoidDuplicates:* The source code should be free of duplicates.
- *AvoidManyGetters:* The ratio between getter methods and the total number of fields should not exceed a certain threshold.
- *AvoidManySetters:* The ratio between setter methods and the total number of fields should not exceed a certain threshold.
- *AvoidMassiveCommentsInCode:* A method should not have too many comment lines in the code. The method documentation (API-documentation) and blank lines are not considered.
- *AvoidNonCohesiveImplementations:* A class should not have sets of methods that are not related to each other. Related means that they use/change the same set of fields or are connected by method calls.
- *AvoidNonCohesivePackages:* A package should be as cohesive as possible, i.e., packages should not contain independent groups of classes.
- *AvoidPackageCycles:* The usage of classes and interfaces in different packages should not create a package cycle.
- *AvoidProtectedFields:* A class should not have protected fields.
- *AvoidPublicFields:* A class should not have public fields.
- *AvoidPublicStaticFields:* A class should not have global variables, i.e., no public static fields.
- *AvoidReturningDataFromCommands:* A public method identified as command method should not return any kind of data regardless whether the data is related to the internal state of the object or not.
- *AvoidRuntimeTypeIdentification:* Type checks of objects, i.e., use of instanceof operator in Java or the typed operator as well as the dynamic_cast operator in C++, should be avoided.
- *AvoidSettersForHeavilyUsedFields:* A class should not have setter methods for a private field that is heavily used. A field is heavily used if it is read or written in more than five methods including getter and setter methods.
- *AvoidSimilarAbstractions:* Different types should not represent a similar structure or behavior. Two classes have a similar structure if the fields with same type and a similar name (word stem) overlap by a particular percentage. Two classes have a similar behavior if methods with the same return type and parameter types as well as similar name (word stem) overlap by a particular percentage.
- *AvoidSimilarNamesForDifferentDesignElements:* Design elements of different kinds should not have similar names, i.e., a package name should not be similar to a class name.
- *AvoidSimilarNamesForSameDesignElements:* Design elements of same kind (e.g., (abstract) classes, interfaces, or packages) should not have similar names.
- *AvoidStronglyCoupledPackages:* A package should not heavily depend on other packages. Therefore, packages containing many classes that depend on types from other packages should be avoided.
- *AvoidUncheckedParametersOfSetters:* A field should only be set by a method parameter that is checked before. This can be verified by checking whether setting the field by a parameter of a (set)-method is always (or at least often) guarded by checks.

- *CheckUnsuitableFunctionalityOfClass:* The methods of a class should be used as whole and not only (small) sets of them. If clients typically use only parts of the methods provided by a class, the functionality of the providing class seems not fit the needs of the clients.
- *CheckUnusedSupertypes:* If clients (not subclasses!) of a class use only the public methods of the current subtype and do not use any methods of a supertype, then there is not a true is-a relationship between the class (subtype) and its supertype.
- *DocumentInterfaces:* An interface must have an API-documentation for the interface declaration and each method signature.
- *DocumentPublicClasses:* A public class and public struct (in C++) must have an API-documentation, i.e., comments above the declaration or the definition of the particular entity.
- *DocumentPublicMethods:* A public method within a public class must have an API documentation, i.e., comments above the method declaration or the definition of the particular entity.
- *DontReturnMutableCollectionsOrArrays:* A method should not return an array or an instance of a collection type. A method is excluded from this rule when the return value is immutable or cloned before.
- *DontReturnUninvolvedDataFromCommands:* A command method that changes the state of the object or class cannot return data that is not related to the change.
- *ProvideInterfaceForClass:* A public class should provide an interface that is used as type for variables and parameters. Classes that provide only access to static members are excluded by this rule.
- *UseAbstractions:* A package should provide a sufficient number of abstract classes and interfaces expressed by the ratio between abstract and concrete types.
- *UseCompositionNotInheritance:* A class should use composition instead of inheritance when the class accesses only public members from the particular superclass. Interfaces and abstract classes are excluded by this rule.
- *UseInterfacesAsReturnType:* If the return type of a method is not a base data type, it should be the interface or the abstract superclass of the class.
- *UseInterfacesIfPossible:* Use an interface for variable declarations, parameter definitions, or return types instead of a public class when the interface provides all methods that are needed.

Appendix B. Suggested best practices

- *UseImmutableObjects:* Use immutable objects as far as possible, i.e., field initialization in constructors and const/final members.
- *DontExposeInternalStructureOfClass:* Internal design decisions should not be recognizable to clients by means of the class interface (method naming, granularity, parameters, and return values). For example, a client should not know if a point is stored as polar coordinates or Cartesian coordinates (e.g., by using a “getX()” and “getY()” method for Cartesian coordinates, but a “calculateRadius()” and “calculatePolarAngle()” method for polar coordinates).
- *AvoidExposingImplementationDetailsInMethodNames:* The name of a public method should not expose the implementation details.
- *MakeClassesPublicIfNecessary:* A class or data structure not part of the public interface of a component or package should be private.
- *CheckInterrelatednessofLayer:* It should be explicitly specified which layers are allowed to interact with the other layers.
- *AvoidReturningCollectionsArrays:* Avoid returning a collection of arrays or an array of collections.
- *AvoidDuplicationOfStateInformation:* The state of a part of an application should be represented in a single class and only be retrieved by other classes from there without permanently duplicating the state information in the local fields of the client classes.
- *CheckMethodUsageDynamically:* Check during runtime whether there are groups of methods used in the different life-time phases of an object.
- *AvoidUnbalancedInheritanceAndDelegationHierarchies:* Factorize complex implementation details into separate classes that are integrated by using delegation. Additionally, avoid complex inheritance structures in the delegated classes.
- *AvoidUnrelatedFields:* A class should not have fields with unrelated semantics.
- *AvoidStaticMethods:* A class should not define static methods, except for special cases such as implementing the singleton pattern.
- *AvoidNonCohesiveInterfaces:* An interface should not have sets of interface methods that are not related to each other. Related means that an interface is fully implemented by its implementations without providing default methods.
- *AvoidLargeObjects:* Avoid inheritance when inheritance results in objects with a huge number of fields (aggregated by all classes in the inheritance tree).
- *UseCoherentNaming:* Groups of classes belonging together in a package should be named coherently (i.e., indicate their belonging to this group by a stable part of the class name).
- *AvoidSharedClassesInSubPackages:* A shared class, which is used by classes in different packages and is no generic API, should be in a super package related to the classes (clients) using it.
- *AvoidSimplyDependenciesAcrossMultiplePackages:* A class depending on just another class should be in a sub-package of the dependent class.
- *AvoidReturningContainerObjectsFromCommands:* If the state of an object is changed by a command, the returned data should be at the finest possible level of granularity (e.g., if only one item in a list is changed, the whole list should not be returned).
- *AvoidMutableFieldsWhenPossible:* A field should be private and final when it is never changed.
- *AvoidPretendedObjectStates:* The state of an object using its properties should never be pretended (e.g., the return type of a query should represent the real object state).
- *UseMeaningfulVariableNames:* Variable names must be meaningful.
- *CheckPrivateMethodUsage:* A private method cannot be reused outside the class and might lead to duplications.
- *CheckGroupingOfUtilityClasses:* Utility classes should be grouped, ideally in the same package.
- *AvoidSameInformationInDifferentArtifacts:* Decisions made in the design process should not repeatedly be described in other artifacts and at other levels; for example, the reason for splitting a component described in the architecture should not be repeated in the design of the component.
- *AvoidDeadCode:* The unused code in a method should be removed.
- *AvoidNonCohesivePackageImplementations:* A package should be as cohesive as possible (i.e., packages should not contain independent groups of classes).
- *UseStrictLayering:* By grouping the libraries into layers, you can define where those libraries intersect. Low-level layers must not use functionality from upper-level layers.
- *KeepInterfaceImplementationsTogether:* Group classes in the same package that share the same interface.
- *AvoidReferencingImplementationPackages:* Avoid dependencies between concrete (implementation) packages but use abstract

packages instead. Within abstract packages, it is only allowed to place interfaces and super classes.
- *AvoidPropertyInjection:* A class property should not be injected from outside the class.
- *UseInterfaceForExternalPackageDependencies:* Use interfaces for access by classes outside the package. Use abstract classes for classes that will be in the same package or library.
- *CheckUnrelatedMethods:* If a method is unrelated to its class, then it should be in another class.
- *AvoidInterfaceInheritance:* Avoid extensive usage of the inheritance mechanism with interfaces since this might lead to unspecific and broad interfaces.

References

Abdeen, H., Sahraoui, H., Shata, O., 2013. How we design interfaces, and how to assess it. In: 29th IEEE International Conference on Software Maintenance (ICSM). IEEE, Eindhoven, Netherlands, pp. 80–89. doi:10.1109/ICSM.2013.19.

Adler, M., Ziglio, E., 1996. Gazing Into the Oracle: The Delphi Method and Its Application to Social Policy and Public Health. Jessica Kingsley Publishers, London, UK.

Briand, L.C., Daly, J.W., Wust, J.K., 1999. A unified framework for coupling measurement in object-oriented systems. IEEE Trans. Softw. Eng. 25 (1), 91–121. doi:10.1109/32.748920.

Briand, L.C., Wüst, J., Lounis, H., 2001. Replicated case studies for investigating quality factors in object-oriented designs. Empirical Softw. Eng. 6 (1), 11–58. doi:10.1023/A:1009815306478.

Bräuer, J., Plösch, R., Saft, M., Körner, C., 2017. Improving object-oriented design quality: apportfolio- and measurement-based approach. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement. ACM, Gothenburg, Sweden, pp. 244–254. doi:10.1145/3143434.3143454.

Bräuer, J., Plösch, R., Saft, M., Körner, C., 2017. A survey on the importance of object-oriented design best practices. In: 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, Vienna, Austria, pp. 27–34. doi:10.1109/SEAA.2017.14.

Charalampidou, S., Ampatzoglou, A., Avgeriou, P., 2014. A process framework for embedded systems engineering. In: 40th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Verona, Italy, pp. 137–140. doi:10.1109/SEAA.2014.58.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493. doi:10.1109/32.295895.

Churcher, N., Frater, S., Huynh, C.P., Irwin, W., 2007. Supporting OO design heuristics. In: 18th Australian Software Engineering Conference (ASWEC). IEEE, Melbourne, Australia, pp. 101–110. doi:10.1109/ASWEC.2007.47.

Coad, P., Yourdon, E., 1991. Object-Oriented Design. Prentice Hall, London, UK.

Dooley, J., 2011. Object-oriented design principles. In: Software Development and Professional Practice. Apress, Berkeley, CA, USA, pp. 115–136. doi:10.1007/978-1-4302-3802-7_10.

Edmunds, H., 2000. Focus Group Research Handbook, 1 McGraw-Hill, Lincolnwood, Ill.; St. Albans.

Hunt, A., Thomas, D., 1999. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley, Boston, MA, USA.

Kontio, J., Bragge, J., Lehtola, L., 2008. The focus group method as an empirical tool in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (Eds.), Guide to Advanced Empirical Software Engineering. Springer London, pp. 93–116. doi:10.1007/978-1-84800-044-5_4.

Linstone, H.A., Turoff, M., 1975. The Delphi Method: Techniques and Applications. Addison-Wesley Pub. Co., Advanced Book Program.

Marinescu, R., 2004. Detection strategies: metrics-based rules for detecting design flaws. In: 20th IEEE International Conference on Software Maintenance (ISCM). IEEE, Chicago, IL, USA, pp. 350–359. doi:10.1109/ICSM.2004.1357820.

Martin, R.C., 1996. Granularity. C++ Report 8 (10), 57–62.

Martin, R.C., 2003. Agile Software Development : Principles, Patterns and Practices. Pearson Education, Upper Saddle River, NJ, USA.

Martin, R.C., 2008. Clean Code: A Handbook of Agile Software Craftsmanship, 1 Prentice Hall, Upper Saddle River, NJ.

Meyer, B., 1997. Object-Oriented Software Construction, 2nd ed Prentice Hall PTR, Upper Saddle River, N.J.

Morgan, D.L., 1997. Focus Groups as Qualitative Research, 16. SAGE Publications.

Okoli, C., Pawlowski, S.D., 2004. The Delphi method as a research tool: an example, design considerations and applications. Inf. Manage. 42 (1), 15–29. doi:10.1016/j.im.2003.11.002.

Plösch, R., Bröuer, J., Körner, C., Saft, M., 2016a. Measuring, assessing and improving software quality based on object-oriented design principles. Open Comput. Sci. 6 (1), 187–207. doi:10.1515/comp-2016-0016.

Plösch, R., Bräuer, J., Körner, C., Saft, M., 2016b. MUSE - Framework for measuring object-oriented design. J. Object Technol. 15 (4), 2:1–29. doi:10.5381/jot.2016.15.4.a2.

Riel, A.J., 1996. Object-Oriented Design Heuristics, 1st Addison-Wesley, Reading, MA, USA.

Samarthyam, G., Suryanarayana, G., Sharma, T., Gupta, S., 2013. MIDAS: adesign quality assessment method for industrial software. In: 35th International Conference on Software Engineering (ICSE). IEEE, San Francisco, CA, USA, pp. 911–920. doi:10.1109/ICSE.2013.6606640.

Schmidt, R.C., 1997. Managing Delphi surveys using nonparametric statistical techniques. Decis. Sci. 28 (3), 763–774. doi:10.1111/j.1540-5915.1997.tb01330.x.

Sharma, T., Samarthyam, G., Suryanarayana, G., 2015. Applying design principles in practice. In: 8th India Software Engineering Conference (ISEC). ACM, Bangalore, India, pp. 200–201. doi:10.1145/2723742.2723764.

Srivastava, S., Kumar, R., 2013. Indirect method to measure software quality using CK-OO suite. In: 2013 International Conference on Intelligent Systems and Signal Processing (ISSP). IEEE, pp. 47–51. doi:10.1109/ISSP.2013.6526872.

Stevenson, J., Wood, M., 2017. Recognising object-oriented software design quality: a practitioner-based questionnaire survey. Softw. Qual. J. 1–45. doi:10.1007/s11219-017-9364-8.

Subramanyam, R., Krishnan, M.S., 2003. Empirical analysis of CK metrics for object-Oriented design complexity: implications for software defects. IEEE Trans. Softw. Eng. 29 (4), 297–310. doi:10.1109/TSE.2003.1191795.

Turney, L., Pocknee, C., 2005. Virtual focus groups: new frontiers in research. Int. J. Qual. Methods 4 (2), 32–43. doi:10.1177/160940690500400203.

Wieringa, R., Moralı, A., 2012. Technical action research as a validation method in information systems design science. In: 7th International Conference on Design Science Research in Information Systems: Advances in Theory and Practice (DESRIST). Springer-Verlag, Berlin, Heidelberg, Las Vegas, NV, USA, pp. 220–238. doi:10.1007/978-3-642-29863-9_17.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer Berlin Heidelberg doi:10.1007/978-3-642-29044-2.

Johannes Bräuer received his Doctoral Degree at the Department of Business Informatics - Software Engineering at the Johannes Kepler Universitiy Linz. His thesis concentrates on measuring and assessing software design based on fundamental design principles. Further, his research interests are in software code quality and technical debt assessment approaches.

Reinhold Pläsch is associate professor for Software Engineering at the Department of Business Informatics - Software Engineering at the Johannes Kepler University Linz. He is interested in source code quality - ranging from basic code quality to quality of embedded and safety critical systems. He is also interested in automatically measuring the object-oriented design quality based on design principles.

Matthias Saft is working at Siemens Corporate Technology on software development related topics. His focus is code and design quality, its measurement, visualization and improvement. A corresponding architectural foundation is obligatory, and likewise considered. Additionally, he is interested in large scale lean and agile development methodologies, and their application in an industrial context.

Christian Körner is *Senior Key Engineer* at Siemens Corporate Technology in Munich. Professional interests are in the area of technical and management methods for *Development Efficiency*. Projects focus in the recent years was on developing and applying artefact based assessment methods for development organisations and automatic evaluation of software (design) quality. Projects range from small project interventions to large research collaborations with international partners.