Survey Report:Importance of Object-Oriented Design Best Practices

Johannes Bräuer and Reinhold Plösch
Department of Business Informatics – Software Engineering
Johannes Kepler University Linz
Linz, Austria
johannes.braeuer@jku.at, reinhold.ploesch@jku.at

***Abstract*—This report describes the major results of our survey related to the importance of design best practices. It first shows a general overview of the survey including some demographic aspects. Afterwards, Table I represents the main result; namely, the design best practices order by their derived importance. The importance is calculated based on the opinions of the participants who were randomly assigned to the design best practice. In average, 138 participants have expressed their opinion for one design best practice. The report continues with an analysis of the open question. Finally, a conclusion and an outlook for future work are given.**

Keywords— design best practices; design quality; design rules;

I. RESULT

A. General Overview

The survey was available from 26th of October until 21st of November 2016. In total, 294 individuals interacted with the survey meaning that the followed the invitation link and hit the start button on the welcome page. Then there was a drop-out of 28 people on the personal information page and 17 people on the background & experience page. A large portion (35) left during the assessment of the first 7 design best practices resulting in 214 successfully finished questionnaires.

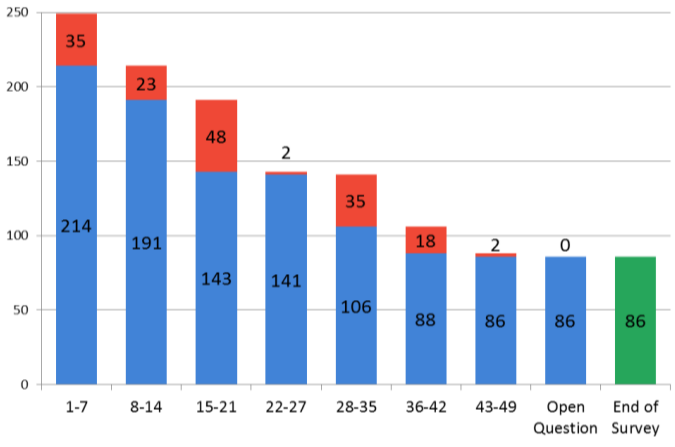

Figure 1. Drop-outs among question blocks

To our surprise, 52.9% of the 214 participants were acquired by posts in social networking platforms. Of course, the posts where placed in topic-related discussion groups and forums, but the number of people who were willing to contribute in this survey is high, compared to previous surveys we conducted.

The design of the survey provided some degree of freedom for the participants and they could decide to end the survey after each assessment block. The analysis of the drop-out rate regarding this degree of freedom is shown in Figure 1. According to the figure and mentioned before, 214 participants expressed their opinion for at least 7 design best practices; afterwards, 23 quit the survey. The drop-out rates between the assessment of the second and third as well as the fourth and fifth block are conspicuous but explainable. At these points the questionnaire presented an image and a short text that showed appreciation for the already invested effort. Besides, the participants had the chance to skip the rest of the survey at these stages. Astonishingly, 86 participants finished the entire survey meaning that they continued until the end of the survey.

Reasons for the high rate of completely finished surveys could come from three aspects. First, the participants were interested in the topic and eager to learn more about object-oriented design practices. Second, the length and duration of assessing a block of design best practices were adequately defined and the participants did not get bored. Third, the idea of giving the participant the power of finishing the survey based on their decision increased the eager to finish the entire questionnaire. While these three reasons cannot be further investigated, it is definitely an interesting aspect concerning qualitative and quantitative research methods.

B. Personal Information

The first question in this block asked about the current affiliation of the participant. Figure 2 shows the distribution of participants from academic organizations, self-employment, and companies differentiated by the number of employees.

According to this diagram, there is a well-balanced distribution between all affiliations ranging from 10 self-employed participants to 33 participants working at a company with more than 1.000 employees. However, there is a strong participation from industry with 176 opinions with a practical point of view rather than a theoretical one.

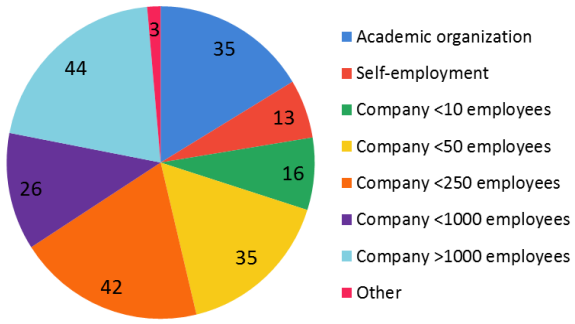

Figure 2. Affiliations of participants

The distribution of participants among the job roles shows that many software engineers and architects expressed their opinion. In more detail, 93 software engineers and 45 software architects – which are representing 64% of the total quantity – successfully finished the survey. Figure 3 depicts the number of participants for each job role. The twelve participants that selected the option *other* noted following roles: Trainer, Teaching, Technical Director, Managing Director (technical), CTO, Director of Institute, Founder, Tech Lead, Software Craftsman, Technical Manager, Managing Director, and Systems Engineer.

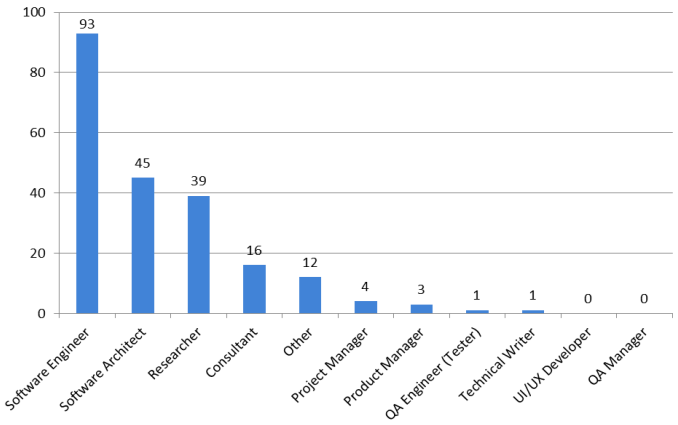

Figure 3. Distribution of job roles

C. Background and Experience

The second question block focused on background information and expertise of the participant in programming languages. The analysis of the domains – in which the participants are working in – highlights two disciplines leading the field. The top business domain is the development of web/service-oriented systems (66) followed by business information systems (56). Figure 4 shows the distribution among the other domains. For the sake of completeness, the twelve participants that selected the option *other* noted following business domains: User Interface, Desktop Applications, CAD/CAM, Human Computer Interaction, Language Implementation, Desktop Applications, Information Systems, Virtual Machines, Tools for Desktop Systems, Compilers, GUI Systems, Finance, CAD System, Healthcare, Nuclear Reactors, Manufacturing Applications.

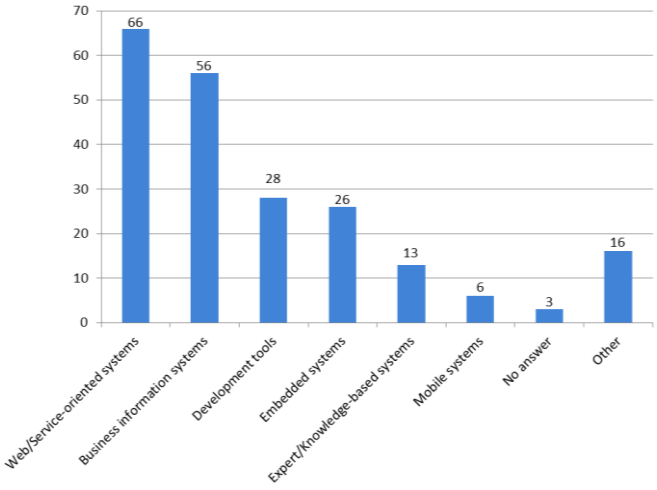

Figure 2. Distribution of business domains

In order to filter the answers and to conduct comparisons among programming languages, we asked the participants to assess their self-appraisal regarding the expertise in a particular object-oriented programming language. The data analysis shows that many participants have top and good experience in Java (125) while C# (80) and C/C++ (84) are less dominant. Nevertheless, there is still a representative group of C# and C/C++ folks with top and good experience therein. Typically, participants have e.g. a top experience in Java and a moderate experience in C++, Figure 5 depicts the experience of the participants. For each participant we consider the highest experience level given regardless of the programing language.

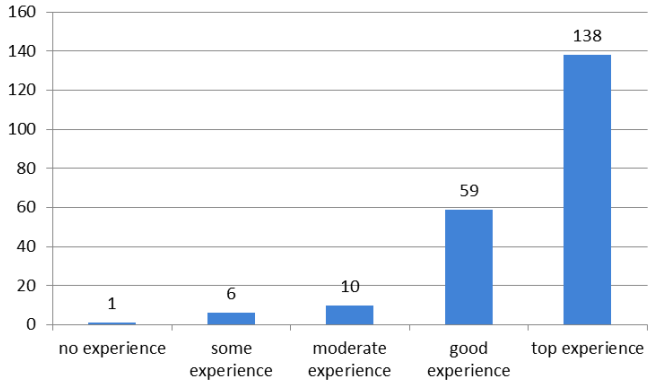

Figure 3. Accumulated programming skills in Java, C# and C++

In addition to the programming languages, a question asked about the self-appraisal regarding object-oriented programming skills. Interestingly, a strong majority assessed this aspect with good or top expertise. The exact distribution of the self-appraisals is shown in Figure 6. This analysis strengthened the representation of our data base and addresses the threat to internal validity as we can show that the participants are familiar with object-oriented concepts and can assess object-oriented design best practices at least to some degree.

Finally, the last question in this block focused on the education of object-oriented programming and software design. While the education at universities – or higher education –

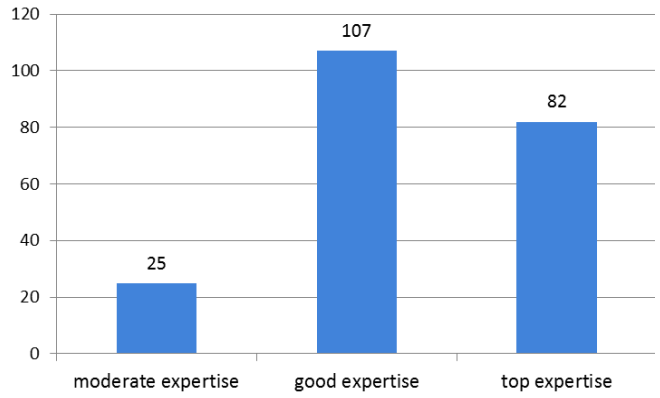

Figure 4. Expertise in oo-programming

plays an important role in teaching oo-programming, almost the same amount of participants said that self-study is another way of gaining skills in this topic. The third place is occupied by on-the-job trainings followed by code review sessions with colleagues. At the end of this list are dedicated workshops.

D. Design Best Practices in Depth

Considering the opinions of all 214 participants, we calculated an average assessment for all 49 design best practices regarding their importance. This calculation uses an equally increasing weight from very unimportant to very important meaning that the assessment of very unimportant is weighted with 1 and very important with 5. As a result, we can show that 5 rules are considered as very important, 21 as important, 12 as moderate important and 11 as unimportant. The exact details are depicted in Table I.

Additionally, this table shows the standard deviation and skew of opinions for each design best practice. The standard deviation ranges from 0.79 to 1.24. A low standard deviation means that the participants agree in their opinions, while a high value stands for disagreement. The skew – as an indicator for the asymmetry of the answers about the mean – shows the tendency to either very important or very unimportant. For this discussion, the skew is multiplied with -1 in order to better express its meaning. Thus, a positive value expresses an important design best practice, while a negative value represents a tendency to unimportance.

The average of all opinions for one design best practice and the standard deviation were used to calculate an importance range for (almost) each design best practice. The only rule that does not provide a range is *AvoidDuplicates* because there is a general agreement on its very high importance. For all the others, the range is shown in the last column of Table I and provides some degree of freedom when using the rules, for instance, to assess a software design. Hints for up- or down grading various design best practices are discussed below.

TABLE I. DESIGN BEST PRACTICES ORDERED BY IMPORTANCE

	Average	Importance	Standard Deviation	Skew * (-1)	Range
AvoidDuplicates	4.42	vh	0.79	1.74	vh
AvoidUsingSubtypesInSuper.	4.27	vh	0.94	1.11	h - vh
AvoidPackageCycles	4.27	vh	0.94	1.43	h - vh
AvoidCommandsInQueryM.	4.23	vh	0.93	0.94	h - vh
AvoidPublicFields	4.09	vh	1.09	1.22	h - vh
DocumentInterfaces	3.96	h	1.07	0.95	m - vh
AvoidLongParameterLists	3.96	h	0.85	0.82	h - vh
UseInterfacesIfPossible	3.96	h	1.08	0.99	m - vh
AvoidStronglyCoupledPack.	3.81	h	0.92	0.41	m - vh
AvoidNonCohesiveImpleme.	3.80	h	1.05	0.78	m - vh
AvoidUnusedClasses	3.80	h	1.06	0.76	m - vh
DontReturnUninvolvedData	3.80	h	0.92	0.72	m - vh
AvoidNonCohesivePackages	3.79	h	0.93	0.51	m - vh
DocumentPublicMethods	3.77	h	1.11	0.89	m - vh
UseCompositionNotInherita.	3.76	h	1.08	0.60	m - vh
DocumentPublicClasses	3.76	h	1.12	0.68	m - vh
AvoidPublicStaticFields	3.74	h	1.15	0.63	m - vh
AvoidDiamondInheritanceSt.	3.68	h	0.95	0.45	m - vh
AvoidLongMethods	3.64	h	1.12	0.44	m - vh
AvoidSimilarNamesForDiff.	3.64	h	1.04	0.35	m - vh
AvoidUnusedAbstractions	3.63	h	1.19	0.61	m - vh
CheckUnsuitableFunctionality	3.61	h	0.95	0.54	m - vh
AvoidSimilarAbstractions	3.60	h	0.94	0.49	m - vh
DocumentPackages	3.58	h	1.19	0.55	l - vh
UseInterfacesAsReturnType	3.54	h	1.20	0.40	l - vh
AvoidUncheckedParameter.	3.52	h	1.02	0.45	h - vh
AvoidSimilarNamesForSame.	3.49	m	1.03	0.39	m - h
CheckObjectInstantiatiosByN.	3.45	m	1.07	0.26	l - h
AvoidRepetitionOfPackage.	3.44	m	1.05	0.48	l - h
ProvideInterfaceForClass	3.34	m	1.12	0.33	l - h
AvoidRuntimeTypeIdentifica.	3.32	m	1.07	0.49	l - h
AvoidDirectObjectInstantiati.	3.32	m	1.07	0.41	l - h
CheckUnusedSupertypes	3.32	m	0.91	0.28	l - h
AbstractPackagesShouldNot.	3.31	m	1.03	0.33	l - h
DontReturnMutableCollectio.	3.31	m	1.08	0.23	l - h
AvoidMassiveCommentsInC.	3.24	m	1.20	0.19	l - h
AvoidReturningDataFromCo.	3.14	m	1.05	0.21	l - h
UseAbstractions	3.02	m	1.24	0.12	vl - h
CheckUsageOfNonFullyQual.	2.97	l	0.96	0.09	l - m
AvoidManySetter	2.96	l	1.05	-0.24	vl - m
AvoidHighNumberOfSetters	2.92	l	0.96	0.07	vl - m
AvoidConcretePackage	2.92	l	0.90	0.07	vl - m
AvoidSettersForHeavilyUsed.	2.86	l	1.16	-0.14	vl - m
AvoidAbstractClassesWithO.	2.85	l	1.07	-0.17	vl - m
DontInstantiateImplementatio.	2.85	l	1.08	-0.20	vl - m
AvoidManyGetters	2.68	l	1.10	-0.43	vl - m
AvoidProtectedFields	2.67	l	1.11	-0.39	vl - m
CheckDegradedPackageStru.	2.62	l	0.94	-0.12	vl - m
AvoidManyTinyMethods	2.60	l	1.02	-0.50	vl - m

E. Qualitative Analysis of Open Question

The last question of the survey was an open and optional question asking about object-oriented design aspects that are not covered by the questionnaire. In total, 47 participants used this question to communicate remarks and comments. For analyzing this open question, we first checked the answers for their meaningfulness and then applied qualitative content analysis techniques proposed by Mayring [1]. Thus, we conducted an inductive category development while working through the answers. The derived categories that were used to code all answers are shown below:

- Design Principle in a broader sense
- Design Principle in a narrow sense
- SOLID
- Context dependent
- Testing related
- Suggestion for improvement

In three answers, there is a reference to design principles in a broader sense like abstraction or encapsulation. One participant mentioned that abstraction is well covered in the questionnaire while a second answer uses the principle of encapsulation for a justification of a statement.

Next to the design principles in a broader sense, participants also mentioned design principles in a narrow sense as well as the SOLID principles. Design principles in a narrow sense are more specific and provide good guidance for building high-quality software design [2], [3]. (For more details on this kind of principles, we lengthy discussed the term and its distinction to design principles in a broader sense in Plösch et al. [4].) The acronym of SOLID stands for five specifically selected design principles that are the: single responsibility, open/closed, Liskov substitution, interface segregation, and dependency inversion principle.

In total, 12 people mentioned either design principles in a narrow sense or SOLID in their answer. Some of these remarks link our design best practices to design principles and express the coverage thereof. For example, one participant was missing questions related to the open/closed principle and another argues that SOLID is not entirely covered. Despite missing suggestions for improving the coverage of the design principles with additional design best practices, the comments are a good indicator for showing the awareness of design principles within the software engineering community.

Since object-oriented design is per-se context dependent, a few participants struggled with assessing the importance for various design best practices. In fact, 6 answers refer to the difficulty to assess the importance of some practices in this way or that way dependent on the context or purpose of the system. Besides, 7 participants mentioned that testing aspects are missing in the survey. It is interesting to see that testing plays nowadays such an important role in software engineering so that it is worth to get mentioned in an oo-design matter. For the sack of completeness, a couple of participants provided suggestions for improvements that will be considered in future work.

II. Conclusion and Future Work

This survey provides a good understanding of the importance of design best practices we use in our measuring framework MUSE [5]. Based on that result, we will continue to investigate aspects of measuring and assessing design principles due to rule violations. First insights in this research question are proposed in Plösch et al. [4] and Bräuer [6], but a comprehensive validation is still missing.

In order to reveal results regarding the challenge of measuring object-oriented design principles, we are currently planning an investigation based on the focus group research method. This group discussion is conducted remotely, and requires approximately three hours over a time period of three weeks. If you are interested in being part of this focus group, please do not hesitate to contact the authors.

III. References

[1] P. Mayring, “Qualitative Inhaltsanalyse,” in *Handbuch Qualitative Forschung in der Psychologie*, G. Mey and K. Mruck, Eds. VS Verlag für Sozialwissenschaften, 2010, pp. 601–613.

[2] J. Dooley, “Object-Oriented Design Principles,” in *Software Development and Professional Practice*, Apress, 2011, pp. 115–136.

[3] T. Sharma, G. Samarthyam, and G. Suryanarayana, “Applying Design Principles in Practice,” in *Proceedings of the 8th India Software Engineering Conference*, New York, US, 2015, pp. 200–201.

[4] R. Plösch, J. Bräuer, C. Körner, and M. Saft, “Measuring, Assessing and Improving Software Quality based on Object-Oriented Design Principles,” *accepted for publication in Open Computer Science Journal (Nov. 2016)*.

[5] R. Plösch, J. Bräuer, C. Körner, and M. Saft, “MUSE - Framework for Measuring Object-Oriented Design,” *J. Object Technol.*, vol. 15, no. 4, p. 2:1-29, Aug. 2016.

[6] J. Bräuer, “Measuring Object-Oriented Design Principles,” in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lincoln, US, 2015, pp. 882–885.