Guiding Peer-feedback in Learning Software Design using UML

Satrio Adi Rukmono
sar@itb.ac.id
Institut Teknologi Bandung
Bandung, Indonesia
Eindhoven University of Technology
Eindhoven, The Netherlands

Michel R. V. Chaudron
m.r.v.chaudron@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

ABSTRACT

Students find learning to design software challenging. There are often multiple ways to solve a problem, and it is not easy to recognise how well one is doing. Feedback from the lecturer, teaching assistant, or peers may help students learn from their mistakes. In this paper, we study students giving and receiving peer feedback on software design to discover the type of feedback that students find helpful, to provide guidance in giving good feedback, and to learn how students use the feedback they receive to improve their design. We examine data from a software project course for third-year informatics bachelor students. We asked students to give peer feedback and respond to the feedback they received. We discovered that students value i) explicit positive feedback, ii) feedback with specific examples, and iii) separate feedback on syntax and semantics. We present guidelines for stimulating helpful peer feedback and found that students' motivation or seriousness in working with the assignment affects their willingness to incorporate the feedback they received into their design.

CCS CONCEPTS

• **Software and its engineering** → *Unified Modeling Language (UML)*; **Designing software**; • **Social and professional topics** → **Student assessment**; **Software engineering education**.

KEYWORDS

software engineering education, design feedback, UML

ACM Reference Format:
Satrio Adi Rukmono and Michel R. V. Chaudron. 2022. Guiding Peer-feedback in Learning Software Design using UML. In *44nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510456.3514148

1 INTRODUCTION

Students find learning to design software challenging. One reason is that there are often multiple ways of solving a problem instead of one correct solution. Moreover, unlike source code, a software design cannot be 'tested' to validate its correctness. Also, there are multiple perspectives to take into account when designing software [6]. As a result, it could be hard for students to understand each perspective cleanly, thus making it difficult to recognise how well they are doing.

Feedback may help students learn from their mistakes. Feedback is commonly provided by lecturers, teaching assistants (TA), or student peers in a higher education setting. Asking students to provide feedback to their peers put them in a different perspective. This perspective can help them gain a better understanding of software design quality, i.e., the act of giving feedback forces students to think deeper on design (including their own). A concern of novice peer feedback is that it can be useless or even harmful. Therefore, for peer feedback to be meaningful, we need to recognise the type of feedback that students find helpful and guide students to give helpful feedback.

In this paper, we explore the use of peer feedback to increase students' learning of software design. We aim to gain insight into the following research questions:

RQ1 What type of feedback do students find helpful?
RQ2 How to instruct, or guide, students to give good feedback?
RQ3 How do students use the feedback they receive to improve their design?

Overall, we pay special attention to what students learn from feedback in software design.

The remainder of this paper is organised as follows. First, we discuss the background to our research, i.e., what readers are expected to be familiar with, in Section 2. We then discuss other related studies in Section 3. Next, we describe the methodology for the study we conducted in Section 4. Section 5 discusses our findings and Section 6 answers the research questions. We discuss threats to validity in Section 7 and finally, we conclude and suggest future work in Section 8.

2 LEARNING OBJECTIVES

For the remainder of this paper, we assume that the reader is familiar with UML diagrams for describing software designs. In our setting, we do not put the bar very high on the proper use of details of the UML notation. For instance, we do not expect full use of the different kinds of relationship notation provided by class diagrams. Instead, we want students to:

LO1: Come up with a good *decomposition*, i.e., a breakdown of the overall functionality of a system into a set of collaborating components, each of which has a clear, *well-scoped, single responsibility*. To this end, we teach students about role-responsibility stereotypes (as proposed by Wirfs-Brock [11, 12]). We also explain generic design principles, such

ICSE-SEET '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9225-9/22/05…$15.00
https://doi.org/10.1145/3510456.3514148

as layering, coupling and single-responsibility. Even though these students have not been trained much in software architecture design, we believe that inventing a good decomposition is a key skill in becoming good software designers.

LO2: *Abstract* the problem away from implementation details. A typical mistake we see in student works is that they describe their implementation; they use terminologies like "SQL-DB", "Cloud", or "Browser" in their logical view. Instead, we would like them to think in more abstract (and hence generic) terms about the type of functionality such as "User Interface" rather than "Browser". This is also related to thinking in terms of responsibilities.

LO3: Use the notation in a systematic and consistent/uniform way to represent their design. For this, we prioritize consistency of use of notation over strictly following the UML standard, although we subscribe to the value of having the UML standard. *Consistency* applies both within a diagram (e.g., classes should have similar granularity of responsibilities) and inter-diagram (e.g., lifelines in a sequence diagram should reflect classes in the class diagram).

LO4: Use separate views into their design. Students have to include at least a structural/logical view but ideally also some dynamic/process views of their design. They also need to separate these conceptual views from implementation details/physical views.

LO5: Create a good layout. The primary audience of software diagrams is human developers. Hence, the ease of understanding diagrams is vital for their use. In particular, the layout of diagrams is an essential factor for the ease of understanding design diagrams. Moreover, we hypothesize that students with a systematic way of working also create a nice looking layout. Conversely, a sloppy layout is likely a sign of students not paying attention to the systematic ordering of concepts.

LO6: Explain the diagrams using text. In our experience, when asking students to make UML diagrams, they end up neglecting to explain the diagrams. Hence, in our assignment and the feedback guidelines, we explicitly guide the students to add sufficient explanations to their diagrams.

3 RELATED WORK

Karasneh et al. examined the difference in quality assessment of students' UML diagrams conducted by themselves, peers, and experts [5]. While the assessments differ significantly, the study found that students and experts use similar features in assessing the diagrams. It shows that peer feedback can be helpful for software engineering students and acts as a basis for our study. In addition, giving feedback to other students' work can also stimulate students to reflect on their own design.

A study conducted by Stikkolorum et al. on difficulties and strategies in designing class diagrams suggests that a good diagram layout may indicate good design [10]. In summary, the study theorised that a good layout helps student understanding their model and provides better insight to evaluators. Our findings agree with this observation and offer more indicators of good design.

Stikkolorum et al. [9] also studied the role of TA in discovering students' difficulties in performing software design and guiding them. This study found that students report difficulties in abstraction and identifying the responsibility of design elements. In addition, TA found that students face difficulties using the (UML) notation correctly, writing report documents, and understanding the task/deliverable itself. In guiding students, TA explains their problems using examples, suggesting directions for improvement, pointing out errors, and suggesting a follow-up meeting after the evaluation.

Jolak et al. [4] found that UML modelling triggers design thinking. They suggested that software designers should not consider modelling costly because it leads to significant thinking about the design and better modularity. This finding validates the use of UML in a software engineering classroom setting.

In a study, Prasad and Iyer [8] asked students to evaluate and find defects in existing software design diagrams. The study provided insight on how students think in assessing design diagrams: students focus primarily on semantic elements in the design and new elements absent in the design and pay less attention to the syntactic elements in the design.

Some researchers attempted to use automation in software design assessment, either as a fully automated grader [1, 2] or as an assistance tool for both students and evaluators [3]. However, in both cases, they require a reference solution from the task maker. This approach cannot work for our studied course, since the projects are from real world cases that do not (yet) have a working solution.

Studies [9] and [10] serve as a motivation for our study. Some of our feedback questions were constructed based on what students and TAs found to be difficult in [9]. Our study is focused on software design phase compared to [10], which covered both analysis and design phases.

4 APPROACH

We examine the data from an undergraduate-level software project course for third-year informatics students. Groups of 5 students were tasked with performing an iterative, real-case software development process. The number of students in each group was decided by the course's lecturers. Students were given lectures in software architecture design, including topics in functional decomposition, the separation of logical and implementation details, and Wirf-Brock's role stereotypes of classes in object-oriented design [11]. Students' grades in prior, relevant courses (such as programming and software engineering courses) were taken into account when dividing them into groups to ensure the uniform distribution of student skills.

In the group project, students delivered their design documents (among other deliverables) that included UML diagrams and their design descriptions in each two-week long iteration. The design must include use case, class, sequence, and deployment diagrams corresponding to the scenario, logical, process, and physical view from Kruchten's 4+1 view model [6], respectively. Students were required to use StarUML [1] for drawing UML diagrams.

After the third iteration, student designs were distributed to their peers. Smaller groups of 2–3 students ("reviewer groups")

[1] https://staruml.io

were then asked to give feedback on the works of two other groups ("project groups"), i.e., each project group received feedback from four reviewer groups. We asked smaller groups to give feedback to avoid project groups assigning the "review task" to individual members, preventing other members to experience the act of giving feedback. However, we still want students to give feedback as pairs to encourage discussions and produce higher-quality feedback compared to individuals.

Finally, the project groups were asked to select the two feedback out of four that they considered most helpful and then respond to the feedback, stressing how they would improve their design in the next iteration based on the feedback. We left how to select the best feedback to each group.

To summarise the numbers, 155 students were enrolled in the course, divided into 31 project groups of 5. Nine project owners (external entities) provided 15 problem domains. Each domain was worked on by 1–3 project groups independently. For giving feedback, each of the 31 groups was divided in two, with each sub-group reviewing the works of two other groups—this reviewer group reviewed at most one other project group that worked on the same problem domain. In reality, one of the 62 reviewer groups failed to give any feedback, causing two project groups to receive feedback only from three reviewer groups. Other than that, a mix-up in the distribution of groups caused one project group to receive feedback from six reviewer groups instead of four, and two other project groups received three instead of four.

Each project group was asked to select two feedback responses; however, not all groups responded to the feedback accordingly. Notably, one project group did not provide any responses, and three only responded to one feedback response. Another project group responded to three feedback responses instead.

We collected the feedback and responses via online forms. This study focuses on the logical view, and as such, the feedback we asked students to provide revolves around class diagrams. The questions in the survey forms can be seen in Tables 1 and 2, respectively. Students were highly encouraged to use English in their design (especially diagram texts), feedback, and comments. However, English was not the primary language in this course, hence the first two questions in the feedback form about the language used.

Table 3 shows the mapping between learning objectives and feedback questions.

With students' responses in hand, we manually analyse the data, especially for free text questions. First, we categorise and quantify the responses to get a high-level view of the situation and select notable data points. Then we dive into the data with qualitative analyses and draw our conclusions.

Table 1: Questions for reviewers.

№	Feedback question	Question type
1	Are the text in diagrams English?	Yes/No
2	Are the text explaining the diagram English?	Yes/No
3	Does the architecture have a good decomposition into sub-functions?	Yes/No
4	Please elaborate on the previous question, especially if you answered 'No'.	Free text
5	Do all components have a clear responsibility?	Yes/No
6	Please elaborate on the previous question, especially if you answered 'No'.	Free text
7	Are the role-stereotypes given for each component correct?	Yes/No
8	Mention explicitly which component(s) you think should have a different role-stereotype.	Free text
9	Are there components that represent implementation concepts rather than conceptual/functionality?	Yes/No
10	Mention explicitly which component(s) you think represent implementation concepts rather than conceptual/functionality.	Free text
11	Additional feedback on the quality of the design	Free text
12	Are class names easy to understand (descriptive of the function of the component)?	Yes/No
13	Are class names specific?	Yes/No
14	Additional feedback on class names	Free text
15	Are method names easy to understand?	Yes/No
16	Are method names specific?	Yes/No
17	Additional feedback on method names	Free text
18	Does the software architecture document (SAD) provide enough description of the system in natural language?	Yes/No
19	Additional feedback on document clarity	Free text
20	How would you rate the visual appearance of the diagram?	Scale 1–5
21	Give your comment on the visual appearance & aesthetics of the diagram	Free text

5 RESULTS

In this section, we discuss the results of our analyses on the feedback and responses. We separate our findings into text language, visual appearances, naming, as well as design quality and document clarity.

On text language

Before discussing our findings regarding the languages used, we want to explain the use of English, which is not a primary language in the course. The primary language, Indonesian, is a relatively young language. Software engineering learning materials including the textbooks used in the program are mostly in English, and there are many terms that cannot be easily translated to Indonesian. Therefore, students are expected to have intermediate-level English proficiency, which is ensured by the required English course that students have to complete in the first year of the bachelor program.

The languages used in the submitted design documents can be seen in Figure 1. The numbers correspond to the answers to feedback questions 1 and 2. Despite encouragement from the lecturers, 23 out of 31 project groups (74.2%) use a language other than English in their UML diagrams. Only six groups (19.3%) use English at all in their design description, i.e., the narration explaining the diagrams. Looking into their diagrams, we find that some groups

Table 2: Questions for project groups aimed to understand their responses to feedback.

№	Response question	Question type
1	Which improvements do you plan to make to your architecture/design based on the feedback?	Free text
2	Do you plan to change anything about the decomposition of your design? Why?	Free text
3	Do you plan to change any responsibilities of the classes in your design? Why?	Free text
4	Do you plan to change anything about the role-stereotypes in your design? Why?	Free text
5	Do you plan to remove/rename any concepts that represented implementation concepts? Why?	Free text
6	Did the feedback help you understand role-stereotypes? Please explain as specifically as possible.	Free text
7	Do you plan to change any names of classes? In what way? Why?	Free text
8	Do you plan to change any names of methods? In what way? Why?	Free text
9	Do you plan to change anything in the explanation in your software architecture document (SAD)?	Free text
10	How would you improve the visual appearance of your class diagram?	Free text
11	Was the feedback clear?	Free text
12	Is there any feedback missing that you would like to have received?	Free text

Table 3: The mapping between learning objectives and feedback questions.

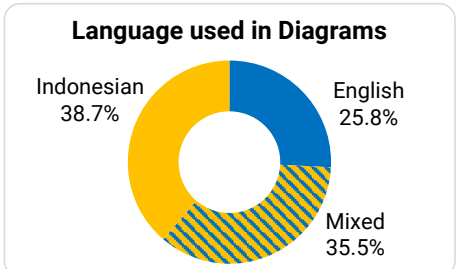
ID	Learning Objective	Related Questions
LO1	Come up with a good decomposition	3, 4
LO2	Abstract the problem away from implementation details	5–10
LO3	Use the notation in a systematic and consistent/uniform way	11–17
LO4	Use separate views into their design	9, 10
LO5	Create a good layout	20, 21
LO6	Explain the diagrams using text	18, 19

mix languages. Further investigation reveals that students found that some application domain-specific terms are hard to translate to English.

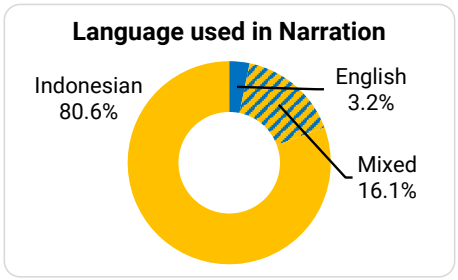
In reaction to the feedback, 10 project groups out of 23 that use language other than English (43.5%) comment that they will change the text into English in the next iteration.

On the visual appearance of diagrams

The reviewers were rather generous in rating the visual appearance of diagrams (feedback question 20) with an average of 4.40 and a standard deviation of 0.71. Figure 2a shows the average rating that each project group received from their reviewers, with each vertical bar representing a project group, sorted by average rating received. Only four groups receive a rating below 4, with one below 3. This project group with the lowest rating on visual appearance did not use the required tool StarUML to create their diagrams.



(a) Text in diagrams



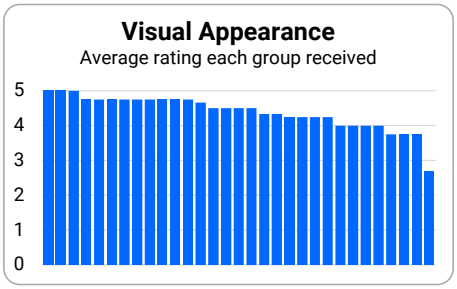
(b) Text explaining diagrams

Figure 1: Language used in diagram text and explanation.

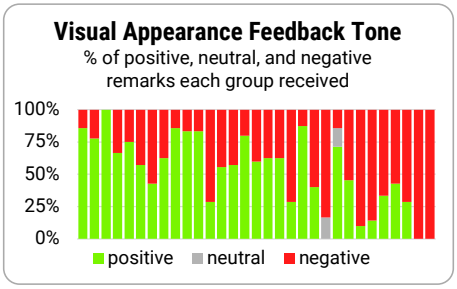
We constructed Figure 2b based on feedback question 21. We categorised the feedback into keywords and organised the keywords into positive, neutral, and negative tones. Similar to Figure 2a, each vertical bar represents a project group, and the groups are ordered according to the rating to illustrate the correlation between the variables. From the feedback tone, we can see that even though the rating was generous, reviewers still gave accompanying negative remarks besides positive ones, not excepting the groups that received a perfect rating of 5. There is a considerable positive correlation of 0.62 between the positive feedback tone and the rating that project groups received, and a similar negative correlation of -0.61 between the negative feedback tones and the rating. These correlations indicate the reviewer's consistency in giving feedback.

Looking into the keywords, most positive remarks are non-specific, e.g., *good*, *neat*, and *visually pleasing*, without describing how the diagram is good. Meanwhile, most negative remarks are more specific, including comments on overlapping lines and layout or spacing problems. There are 122 positive remarks, 105 negative remarks, and two neutral remarks given by 61 reviewer groups to 31 project groups. The details of these keywords can be seen in Figure 2c.

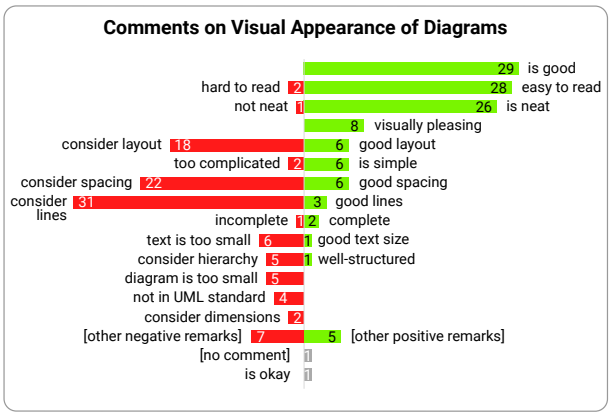
Responding to the feedback on the visual appearance of diagrams, 21 of 31 project groups promised that they would try to improve visual appearance in the next iteration. Twelve of them will only make one kind of change, five will make three kinds of



(a) Average rating



(b) Feedback tone

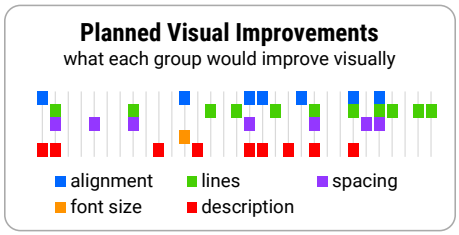


(c) Feedback keywords

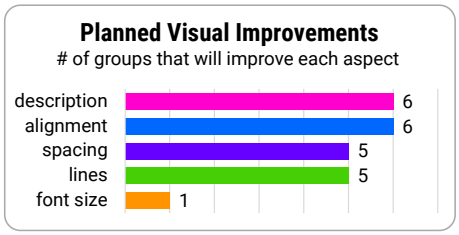
Figure 2: Feedback on visual appearance of diagrams.

change, and the rest will make two kinds of change (Figure 3a). The changes they planned to make were mainly consistent with their feedback; however, it is interesting to note that some groups with the highest rating plan to make more improvements than those with the smallest rating points. We surmise that this relates to how serious the groups were in doing the assignment. Groups with higher intrinsic motivation tend to receive higher ratings and are willing to improve their works even further. In comparison, those not working on the tasks seriously tend to receive lower ratings and are unmotivated to spend more time for improvements regardless of the feedback. Nevertheless, there is a slight positive correlation (0.36) between the number of negative remarks that a group received and the number of improvements they planned to make.

Looking at the aspects of improvements (Figure 3b), the most popular planned improvement is on *diagram description*—which, curiously, does not fit into the category of *visual appearance*, and on diagram alignment. Other aspects include spacing, lines, and font size.



(a) Planned improvements



(b) Aspects of improvements

Figure 3: Responses to feedback on visual appearance.

On naming

Most reviewers positively marked class and method names as easy to understand and specific enough, with 87% to 93% respondents saying 'yes' to feedback questions 12, 13, 15, and 16 (Figure 4). The percentage of positive, neutral, and negative remarks that each group received on class and method naming can be seen in Figure 5a and Figure 6a. The diagrams are sorted by the number of negative remarks each group received in the respective feedback question.

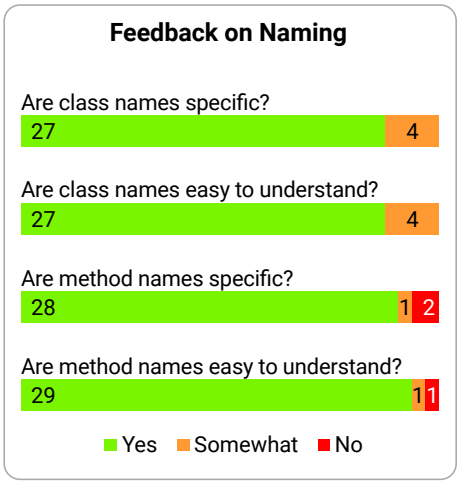
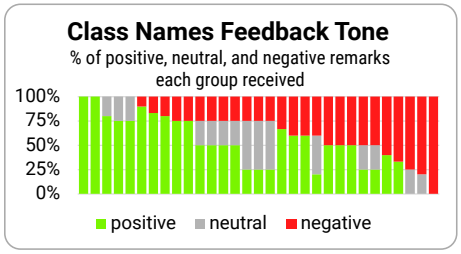
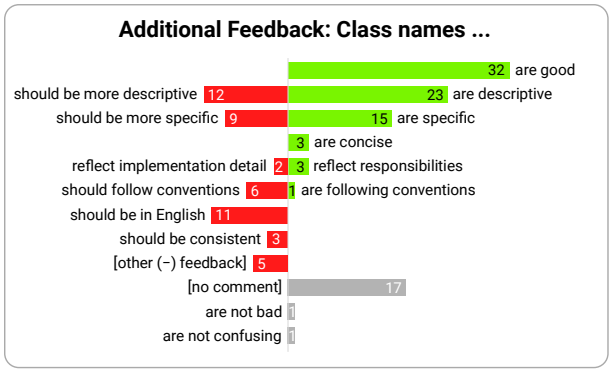


Figure 4: Feedback on class and method names.

When asked to give additional feedback (feedback questions 14 and 17), most negative remarks stated that class and method names need to be more specific and that some classes and methods need renaming for language consistency. To a lesser extent, some suggested following naming conventions and use proper levels of abstraction in names. We organised the keywords in feedback on names similar to what we did with the feedback on visual appearance. The resulting categorisation can be seen in Figure 5b and Figure 6b. An interesting finding from the data is that while there are significantly more reviewers that give positive remarks, the "vocabulary" for negative remarks is more "colourful"—in other words: negative remarks are more nuanced than positive ones.



(a) Feedback tone



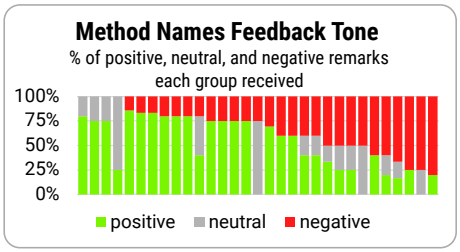
(b) Feedback keywords

Figure 5: Feedback on class names.

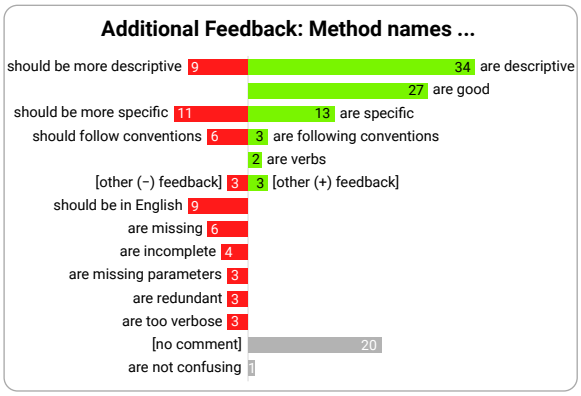
There is a remarkable correlation of -0.55 between the number of negative remarks a group received on class names and the rating they received on the visual appearance of diagrams. Once again, this correlation may reflect the students' seriousness in carrying out software design tasks.

Another notable finding is that some project groups receive conflicting remarks from different reviewers. For example, one reviewer mentioned that group X's class names are descriptive, while another reviewer commented the opposite. However, a closer inspection reveals that the negative remarks usually only apply to a few classes in such occurrences. In contrast, the positive ones apply to the whole design in general.

After being presented with the feedback, many project groups comment that they would revise some class (48%) and method names (55%) even when most groups received positive feedback.



(a) Feedback tone



(b) Feedback keywords

Figure 6: Feedback on method names.

On design quality and document clarity

The design quality is reflected by feedback questions 3–11, while questions 18–19 concern document clarity. We decided to exclude question 9 because many students misunderstood the question due to language difficulties, evident from the contrast in answers to questions 9 and 10. One reviewer acknowledged this confusion by stating that this question differs from the other Yes/No questions, where a "Yes" in response to this question denotes a negative point towards the design, i.e., the existence of components that reflect implementation concepts is not desirable in the logical view. This misunderstanding means the collected data would reflect neither the reviewers' point of view nor the actual design quality. We, therefore, use question 10 as the sole data for the aspect of *components that represent implementation concepts rather than functionality*.

We use question 11, free-text feedback on the quality of the design, to gain what we consider an overview of a project group's design quality at a high level (Figure 7). The bars in the figure are sorted by the percentage of positive remarks each group received. This order is used to sort the data in subsequent diagrams concerning design quality and document clarity. For simplicity, we consider that the groups on the left-hand side of the diagrams made a higher quality design than those on the right.

Getting into the details, we visualised the feedback from questions 3, 5, and 7, and 18—concerning the quality of decomposition, clarity of classes' responsibilities, correctness of assigned role stereotypes, and the clarity of document text—in Figure 8. The diagram shows a predictable correlation: the better the design quality, the more likely it is that reviewers answer "Yes" to these questions.

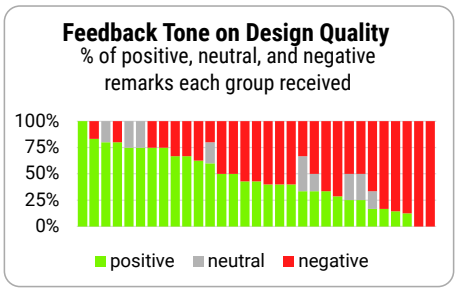


Figure 7: Feedback on overall design quality.

Breaking this down into detail, they correspond to the charts in Figures 9 to 10. Figure 10 shows the existence of incorrect role stereotypes and components representing implementation details in logical view. Again, there are visible correlations between the quality of the design with feedback tone in the quality of decomposition (0.54), the clarity of classes' responsibilities (0.72), the number of "wrong" role stereotypes (-0.39), and the existence of implementation concepts in logical view (-0.38). However, the correlation with document text clarity is more subtle at 0.20.

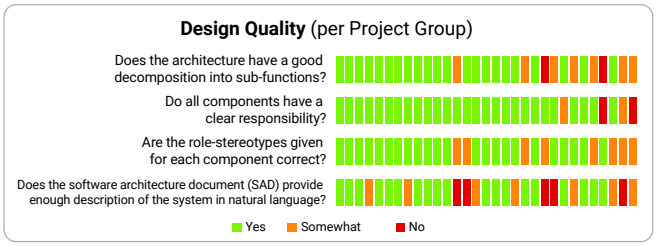


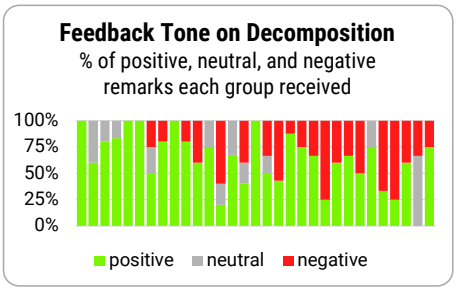
Figure 8: More feedback on the aspects of design quality and document clarity.

Figure 11 depicts how project groups responded to the feedback on design quality. The relation between the variables can again be seen: groups towards the right-hand side of the diagram are more likely to plan revision in each aspect.

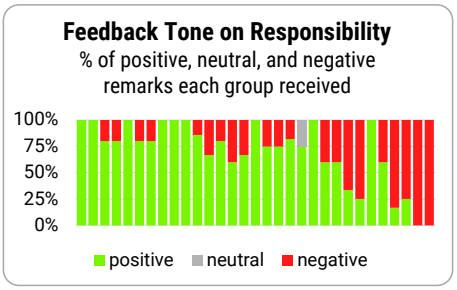
In the response text, students tend to ignore comments on the existence of implementation detail in the logical view. This is to be expected due to the confusion surrounding the meaning of the feedback question, as discussed earlier.

Some students discovered that classes could have more than one role stereotype. Indeed, in real-life cases, it is possible for classes to exhibit the traits of two roles, as Wirfs-Brock's examples "information holders that compute" or "service providers that cache information" [11]. However, this phenomenon in a classroom setting is likely not caused by deliberate modification to typical stereotypes but rather a lack of deep understanding of role stereotypes. Nevertheless, it is interesting to see students discover this by themselves, enriching their knowledge through peer feedback-and-response activity.

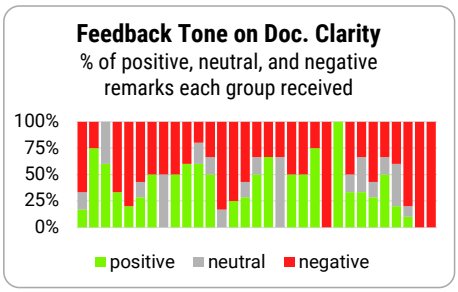
The purpose of writing narration about the design, including diagram explanations, is to convey the architecture to stakeholders. However, some students noted that writing the narration forced them to learn deeper about software design. For example, a student wrote, "[Having to explain the architecture] made me understand more about the role-stereotypes."



(a) Functional Decomposition



(b) Class Responsibilities



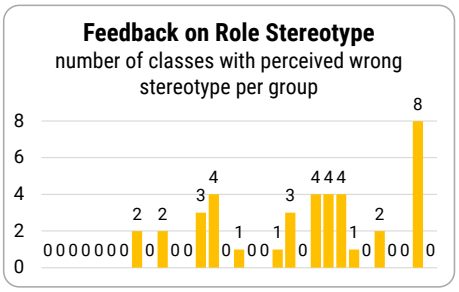
(c) Document Clarity

Figure 9: How each group performs in decomposition, class responsibilities, and document clarity.

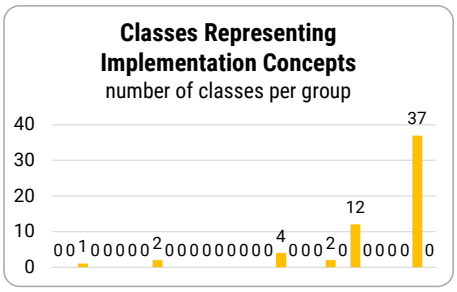
Other findings

In their response to the feedback, some students feel that their feedback is not detailed enough. In addition, it would seem that the feedback questions did not motivate peers enough to take a deep look at their design.

From the correlations among the aspects discussed above, we identified indicators of good software design: visual appearances of diagrams, good class names, clear description of class responsibilities, the use of role stereotypes, and the ability to separate logical view from implementation details. These features are included in Karasneh's list of features both experts and students use when evaluating the quality of UML class diagrams [5] and therefore feasible to use in giving peer feedback.



(a) Role Stereotypes



(b) Components that represent Implementation Concepts

Figure 10: The existence of incorrect role stereotypes and components representing implementation details in logical view.

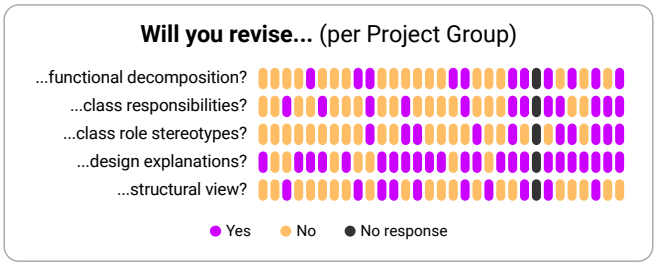


Figure 11: Response to the feedback in design quality and document clarity.

Half the students claimed that peer feedback helped them learn deeper about role stereotypes, as depicted in Figure 12. Elaboration from those who answered 'Yes' to the question include: i) different reviewers provided different point-of-views, ii) reviewers mentioned specific examples of correct and incorrect stereotypes, and iii) the feedback helped to determine role stereotypes from the implicit behaviour of the classes. On the other hand, most that answered 'No' said that i) reviews did not state why their stereotypes are considered correct, and ii) reviews pointed out mistakes but did not suggest how to fix them.

6 DISCUSSION

In this section we address our research questions and provide answers according to our findings.

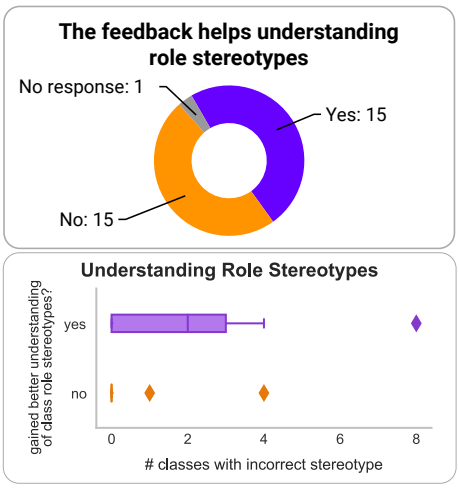


Figure 12: The effect of feedback on learning

6.1 Answering RQ1: What type of feedback do students find helpful?

From both quantitative and qualitative analysis of the feedback and responses, we summarised some features of peer feedback that students find helpful.

Explicit positive feedback. Our feedback questions did not encourage elaboration of positive feedback, i.e., what reviewers find remarkable about the design. The problem with this is that when reviewers do not provide negative remarks, designers cannot be sure if it means their work is excellent or so-so. In other words, a lack of negative feedback is not equal to positive feedback. Therefore, we encourage evaluators (lecturers or TA) and peers to mention what the group did great when giving feedback explicitly.

This significance of positive feedback is consistent with the findings of various studies in the field of education. Paterson et al. [7] provided an extensive review of such studies, noting that positive feedback makes students enthused and gives them a sense of achievement. A balanced positive and constructive feedback also motivates students to improve the quality of their work.

Feedback with specific examples. A student expressed clearly that "[it] would also be more helpful if [reviewers] have given more specific examples along with their feedback to clear up some confusions." Another student mentioned that the feedback they received was clear "because [the reviewers] give us some example in which part we're bad (sic)." We thus implore reviewers to provide specific instances of design issues when giving feedback.

Separate feedback on syntax and semantics. We did not explicitly separate feedback questions into syntactic and semantic aspects. As a result, some reviewers did not elaborate on the semantic soundness of a group's design, leaving the designer clueless. However, relying on student peers to evaluate semantic elements may not be the best choice since they do not yet possess the required experience and insight to give valuable comments.

6.2 Answering RQ2: How to guide students to give feedback?

Here are our suggestions for stimulating helpful peer feedback on software design.

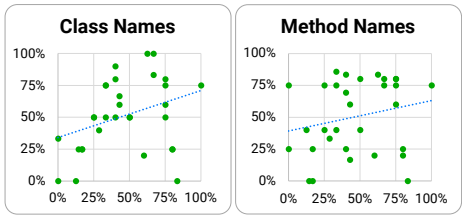
Feedback forms must encourage both positive and negative feedback. For example, in the aspect of class names, instead of simply asking whether class names are easy to understand, a feedback form may ask how many are easy to understand and how many are not.

Feedback forms must explicitly ask for specific examples. Continuing the example of class names aspect, when saying that some class names are not descriptive, reviewers must be required to mention which classes have non-descriptive names. Furthermore, reviewers can be asked to provide counterexample, e.g., "What would be a more descriptive class name?"

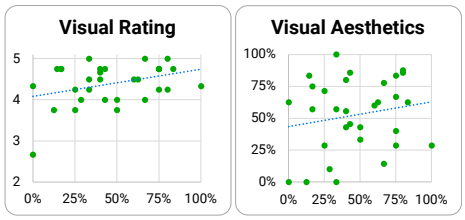
Vocabularies we learned from this study can be used to refine the questions in feedback forms. We extracted keywords from reviewers' comments that can be used to ask more specific questions in feedback forms. For example, in the visual appearance of diagrams, feedback forms can contain specific questions asking peers to rate the layout, spacing, line placements, and text size. For class and method names, we can ask whether the names are specific, descriptive, concise, consistent, reflecting responsibilities instead of implementation concepts, and following naming conventions. In addition, we can also ask whether class names use nouns and method names use verbs. These specific questions may help reviewers navigate the diagrams now that they have a concrete idea of what to expect.

Feedback forms can use some quantifiable design aspects to approximate design quality. We mentioned that peers might not be able to assess the quality of semantic elements reliably. However, the correlations we discussed above can be used to approximate the quality of a software design. Of note, rather than a direct cause-and-effect, the correlation may reflect the students' seriousness in carrying out the software design assignment.

Figure 13 illustrates how feedback tones in several aspects correlate with the overall design quality. In all charts, the x-axes stand for the percentage of positive remarks in question 11 (quality of the design), while the y-axis represent the percentage of positive remarks in each aspect. An exception to this is the chart for visual rating (Figure 13b left) in which the y-axis represent a 5-point star rating, and for role stereotypes (Figure 13c bottom) in which the y-axis represent the number of classes with incorrect role stereotypes. The charts show that *class names*, *decomposition*, *clarity of class responsibilities*, and *correctness of role stereotypes* are good indicators of overall design quality.



(a) "naming" aspects



(b) "visual" aspects

(c) "design quality" aspects

Figure 13: Correlation between perceived overall design quality and positive remarks in several aspects.

6.3 Answering RQ3: How do students use the feedback they receive to improve their design?

Figures 14 and 15 shows the correlation tendencies between the number of negative remarks in each evaluated aspects and the respective improvements students planned to make. In each chart, the x-axis denotes number of negative feedback that each group received and the y-axis denotes a group's willingness to revise their design based on the feedback. We would expect the orange "no" boxes in the plots to be located more to the left (less negative feedback → no plan to revise) and the purple ones ("yes") to the right (more negative feedback → plan to revise).

Students' plans of improvement are mostly as expected; they tend to say that they will improve in aspects that they are lacking. However, we also found that best-performing students planned improvements despite a lack of negative remarks in certain aspects. Conversely, some of those who performed poorly made little to no plan of improvements in some aspects. This may also demonstrate the number of effort students was willing to give in to the assignment.

One particular irregularity can be found in responses regarding the existence of implementation concepts in structural view (Figure 15 bottom). The first to note is that in this chart, the x-axis uses logarithmic scale. This is to make sure that the general trend of the

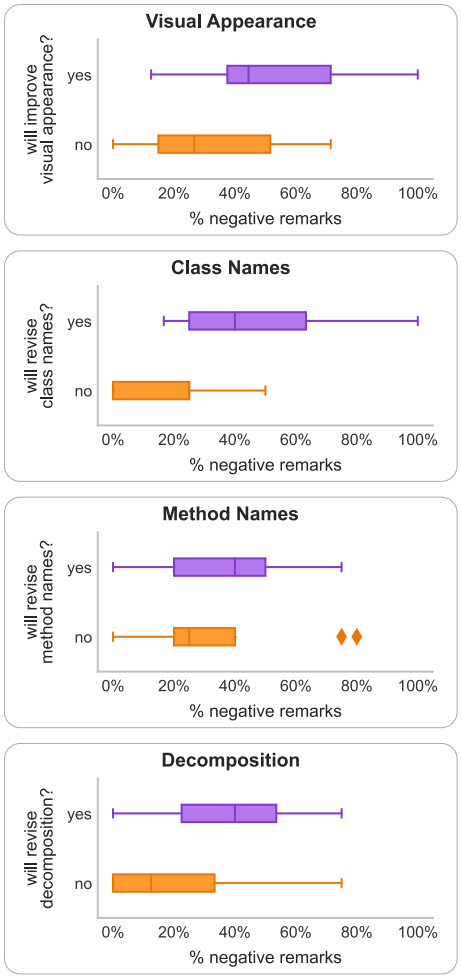


Figure 14: Negative remarks and willingness to improve in the aspects of visual appearance, class- and method naming, and decomposition.

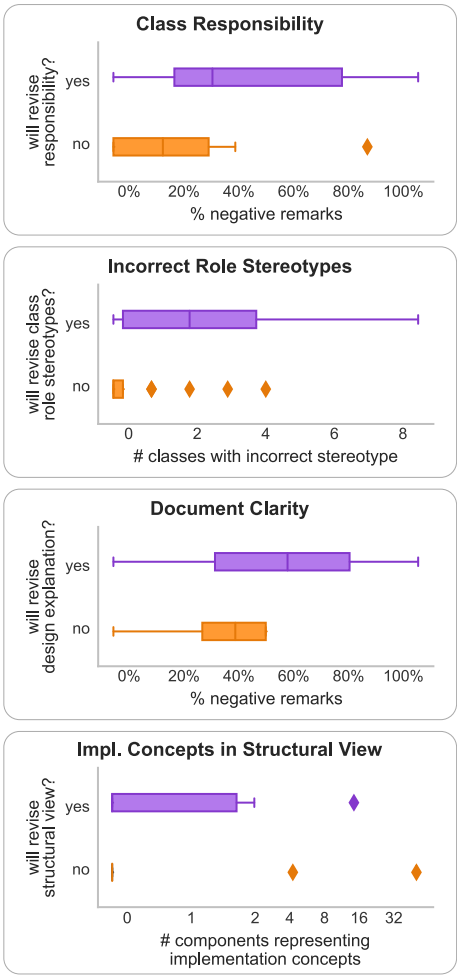


Figure 15: Negative remarks and willingness to improve in the aspects of class responsibilities, role stereotypes, document clarity, and the existence of implementation details in logical view.

chart is visible while including the anomalous data point. in the far right of the chart, there is a group with 37 classes in their diagram that reflects implementation view, which is in fact *all* of the classes in their design. This is because that particular group submitted a class diagram of the classes generated by their coding framework, effectively not representing their design phase.

7 THREATS TO VALIDITY

Construct validity. We did not check all feedback with regards to the design document. In effect, we cannot be sure if the feedback are relevant or actually reflect the quality of the design under review. We asked the students to select the best reviews with the intention to weed out incongruous reviews and minimize the threat to construct validity. There is also a possibility for Hawthorne effect, i.e., students changed their behavior when giving feedback because they know that they are (and consent to) being studied.

Internal validity. We did not specify how students should select the best two reviews out of four. Consequently, we do not know if the selected reviews are indeed the best for supporting the learning objectives.

Reliability. This study involves several free-text questions resulting in hundreds of sentences to analyze. The categorization of such sentences into feedback tones were performed by a single researcher. It would be a better idea to involve other lecturers in deciding the categorization.

8 CONCLUSIONS

This paper reflected on peer feedback-response activity in a software development task for a Software Project course to discover the type of feedback students find helpful and guide them in giving helpful feedback. Our study found the following characteristics for helpful feedback:

i) Explicit positive feedback.
ii) Feedback with specific examples.
iii) Separate feedback on syntax and semantics.

We also suggested the following guidelines to enhance feedback forms for stimulating helpful peer feedback:

i) Encourage the reviewer to provide both positive and negative (constructive) feedback.
ii) Explicitly ask the reviewer to mention specific instances and, when relevant, provide counterexamples.
iii) Use refined feedback questions instead of asking for general comments, e.g., specifically ask about the layout, spacing, and legibility when reviewing diagrams instead of a generic "give your comment on the diagram".
iv) Use quantifiable software design aspects, such as diagram layout and class naming, to approximate overall design quality.

Looking back at our feedback questions, we found questions 7–10 to be working well and already fit some of our proposed guidelines. Question 20 (visual appearance rating) could use more detailed grading criteria and possibly be broken down into finer aspects of the visual appearance. Other questions, mainly in pairs of Yes/No and Free Text questions, need to be improved by incorporating our guidelines. As concrete, non-exhaustive examples, we provide our suggested revision to some of our original feedback questions in Table 4.

Table 4: Examples of suggested feedback questions.

Aspect: class names
Original questions • Are class names easy to understand? • Are class names specific? • Additional feedback on class names
Revised questions • Mention classes with good names and the reason why you find it good. The reason can be one or a combination of the following: descriptive, specific, reflects its responsibilities, follows a certain naming convention. • Mention classes with bad names and the reason why you find it bad. The reason can be one or a combination of the following: non-descriptive, non-specific, reflects implementation concepts, inconsistent across the diagram.
Aspect: visual appearance of the diagram
Original questions • How would you rate the visual appearance of the diagram? • Give your comment on the visual appearance & aesthetics of the diagram
Revised questions Rate the visual appearance of the diagram in the following aspects (1: worst … 4: best) • layout of diagram elements • spacing between elements • line placements • font size • overall visual aesthetics

Future Work

We asked students to select the two most helpful reviews out of four they received in the study. However, we did not explicitly ask why they consider those reviews most helpful. Further studies on this topic should consider asking directly and precisely why and which parts of the review designers find helpful.

We have used and suggested checklists for guiding reviews. Although it might help students give specific feedback, this risks that students think less for themselves, limited to the dimensions the checklist provided. Evaluators need to keep in mind the balance between directedness and ingenuity.

We mentioned the limitation of current approaches to software design assessment automation that require a reference solution. We suggest using the results presented in this paper to design a more general automation tool which, while it may not be able to assess the correctness of the design, can help students by somehow triggering feedback in a mechanism comparable to Bolloju's grader [3]. For instance, an automation review assistant would select classes with potentially problematic names from the design and ask the reviewer to comment on those specific names. For diagram layout, the assistant could ask questions about intersecting lines such as, "Here is a line-crossing. Could it be avoided?"

ACKNOWLEDGMENTS

We express our gratitude towards the lecturers, TA, and students of the observed course for making this study possible and our research assistant Vania Velda for the help in data preprocessing.

REFERENCES

[1] Weiyi Bian, Omar Alam, and Jörg Kienzle. 2019. Automated Grading of Class Diagrams. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019*, Loli Burgueño, Alexander Pretschner, Sebastian Voss, Michel Chaudron, Jörg Kienzle, Markus Völter, Sébastien Gérard, Mansooreh Zahedi, Erwan Bousse, Arend Rensink, Fiona Polack, Gregor Engels, and Gerti Kappel (Eds.). IEEE, 700–709. https://doi.org/10.1109/MODELS-C.2019.00106

[2] Weiyi Bian, Omar Alam, and Jörg Kienzle. 2020. Is automated grading of models effective?: assessing automated grading of class diagrams. In *MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020*, Eugene Syriani, Houari A. Sahraoui, Juan de Lara, and Silvia Abrahão (Eds.). ACM, 365–376. https://doi.org/10.1145/3365438.3410944

[3] Narasimha Bolloju, Ken W. K. Lee, and Probir Kumar Banerjee. 2011. Formative and Summative Assessment of Class Diagrams - Development and Evaluation of a Prototype. (2011), 402–410.

[4] Rodi Jolak, Eric Umuhoza, Truong Ho-Quang, Michel R. V. Chaudron, and Marco Brambilla. 2017. Dissecting Design Effort and Drawing Effort in UML Modeling. In *43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. IEEE Computer Society, 384–391. https://doi.org/10.1109/SEAA.2017.55

[5] Bilal Karasneh, Dave R. Stikkolorum, and Enrique Larios. 2015. Quality Assessment of UML Class Diagrams - A Study Comparing Experts and Students. In *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 29, 2015 (CEUR Workshop Proceedings, Vol. 1555)*, Arnon Sturm and Tony Clark (Eds.). CEUR-WS.org, 55–67. http://ceur-ws.org/Vol-1555/6.pdf

[6] Philippe Kruchten. 1995. The 4+1 View Model of Architecture. *IEEE Softw.* 12, 6 (1995), 42–50. https://doi.org/10.1109/52.469759

[7] Catherine Paterson, Nathan Paterson, William Jackson, and Fiona Work. 2020. What are students' needs and preferences for academic feedback in higher education? A systematic review. *Nurse Education Today* 85 (2020), 104236.

[8] Prajish Prasad and Sridhar Iyer. 2020. How do Graduating Students Evaluate Software Design Diagrams?. In *ICER 2020: International Computing Education Research Conference, Virtual Event, New Zealand, August 10-12, 2020*, Anthony V. Robins, Adon Moskal, Amy J. Ko, and Renée McCauley (Eds.). ACM, 282–290. https://doi.org/10.1145/3372782.3406271

[9] Dave R. Stikkolorum, Francisco Gomes de Oliveira Neto, and Michel R. V. Chaudron. 2018. Evaluating Didactic Approaches used by Teaching Assistants for Software Analysis and Design using UML. In *Proceedings of the 3rd European Conference of Software Engineering Education, ECSEE 2018, Seeon Monastery, Bavaria, Germany, June 14-15, 2018*, Jürgen Mottok (Ed.). ACM, 122–131. https://doi.org/10.1145/3209087.3209107

[10] Dave R. Stikkolorum, Truong Ho-Quang, Bilal Karasneh, and Michel R. V. Chaudron. 2015. Uncovering Students' Common Difficulties and Strategies During a Class Diagram Design Process: an Online Experiment. In *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 29, 2015 (CEUR Workshop Proceedings, Vol. 1555)*, Arnon Sturm and Tony Clark (Eds.). CEUR-WS.org, 29–42. http://ceur-ws.org/Vol-1555/4.pdf

[11] Rebecca Wirfs-Brock. 2006. Characterizing Classes. *IEEE Softw.* 23, 2 (2006), 9–11. https://doi.org/10.1109/MS.2006.43

[12] Rebecca Wirfs-Brock and Alan McKean. 2003. *Object design: roles, responsibilities, and collaborations*. Addison-Wesley Professional.