Measuring the Effectiveness of Feedback on Code of Novice Programmers

Chris Witteveen
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
c.p.t.witteveenl@student.utwente.nl

ABSTRACT

This research investigates the effectiveness of feedback on code quality of novice programmers. Based on an existing model, this research developed different categories for feedback. We developed a definition for effective feedback on code as well as a method to measure the effectiveness of feedback. Using this method feedback on code was analyzed using a small data set. This analysis provides insights into the effectiveness of the different categories developed in this research. The performed analysis is automated into an extension of the atelier program for further research.

Keywords

Feedback, Code Quality, Programming Mistakes

1. INTRODUCTION

Technology is playing an increasingly important role in our lives. People use computers to work, relax or keep in contact with each other. Technology does not only play an important role on a personal level, but also a lot of professions use technology in an increasing amount. The increase in the use of technology generates a new demand for programmers. People in all different kinds of fields are starting to need basic programming skills to operate the current technology. These programmers are often satisfied once their program gives the desired results. However, these programs can often be improved in terms of code quality.

Code quality can be related to a number of aspects [1] including documentation, styling, the flow of the code, or the structure of the code. Issues that are related to flow or structure which can decrease code quality are described as 'code smells' by Fowler[2]. An example of such a code smell is a short variable name which does not give an indication for what the variable is used. These code smells often return in the code of novice programmers.

As described by Hattie en Timperley[3] feedback can be a powerful tool to help learning. Novice programmers, especially students often receive feedback on their code to help them improve their programming abilities. Providing feedback on code is a labour and time intensive task, and not much research has yet been performed to check whether feedback on code smells actually helps improve the code produced by students.

1.1 Background

This research focuses on the study programme Creative Technology (CreaTe) at the University of Twente. Most first year students in this programme are just beginning to code. In the CreaTe programme, tinkering as a means to master the material[4] plays a central role. CreaTe allows students to fully define their own projects, while using the concepts covered in the courses. This means that the student fully owns the problem. There is no reference solution that can be used when assessing the assignment. The produced project has to demonstrate that the student understands and can apply the concepts learned during the course. Not only do the produced projects need to be checked for the concepts learned during the course, also the quality aspects of the projects need to be checked. The produced code needs to be of sufficient quality. Since the students can define their own projects, it is hard to write tests for the code. The code quality thus plays an important role for teaching assistants(TAs) and lecturers to be able to understand the code and provide feedback on it.

The CreaTe programme has recently started using the Atelier platform. The Atelier platform is an new online platform that emphasizes collaboration and the sharing of ideas. It is build for students, TAs, and lecturers involving the CreaTe programme. The aim of the platform is to support the teaching team and to help create a Community of Practice[5]. The platform creates an environment for students and TAs to share code and provide receive feedback on it. Next to the feedback provided by TAs, Atelier also makes use of a tool called Zita to help with providing feedback on code shared by students.

Zita[6] is a static code analysis tool. The tool is custom build by de Man and Fehnker to detect code smells in processing code. The tool is used to automatically generate feedback on code uploaded to the Atelier platform. This helps TAs and lecturers to find code smells more systematically and consistent as this normally requires a lot of work and is prone to errors.

1.2 Problem

Even with Zita generating comments, TAs first need to verify if the generated comments are sound and need to be shown to students. Next to this, TAs still manually check the code for mistakes which are not recognized by Zita. As mentioned before, this is very labour and time intensive and not yet much research has been done to check if this actually helps improve the students programming skills.

In this research a measurement for the effectiveness of feedback on code is composed. This measurement is used on data retrieved from the Atelier platform to provide basic insights into the effectiveness of feedback on code. Next to this, the research also implements this measurement in the form of an extension of the Atelier platform. This extension will be used to provide information about the effectiveness of the feedback to TAs and lecturers. The extension of the Atelier platform enables a longitudinal observational study of the effectiveness of the feedback on code. More data will be collected over a longer period of time allowing for a better analysis.

1.3 Research Questions

This research addresses the following questions:

RQ1 How effective is feedback on code of novice programmers?

- **RQ1.1** Which types can be used for the classification of feedback?
- **RQ1.2** How can the effectiveness of feedback on code be measured?

RQ2 How can the measurement of the effectiveness of feedback be automated for integration in the Atelier project?

- **RQ2.1** How can the measurements be made easily accessible to student assistants and lecturers?

2. RELATED WORK

Some prior research has been done in the field of measuring the effects of feedback on code. This section discusses several papers which investigated issues and the fixing of issues in code. While the different papers all investigated issues in code they differ from this research in some important aspects.

2.1 Code quality issues

Keuning et al.[7] investigated issues regarding code quality in 2.6 million code snapshots written by novice programmers. The code snapshots where taken over a course of 4 weeks. The authors selected 24 code quality issues, divided into 5 categories (Flow, Idiom, Expressions, Decomposition, Modularization), which are relevant for novice programmers. They explored the code snapshots to see how often the selected issues occurred and how often they were fixed. The conclusion of the study is that novice programmers develop programs with a substantial amount of code quality issues which are rarely fixed. This research mainly focused on the occurrence of code smells in programs over time. The authors did not have data regarding feedback of the code.

2.2 Effect of error messages

Kadekar et al.[8] investigated the role of the error message in fixing the problem. They used a think aloud programming exercise to measure the effect on the overall debugging time of the student. For the study they developed three different types of error messages. The Default type, the Link type, and the Example type. The Default type only showed a short error message indicating where the assembler was when the error occurred. The Link type had a more detailed error message with a web link to the online manual for more information. The Example type was a detailed error message with sample code included. The effects of these different types was analyzed. The conclusion of this research was that the Link type helps the students fix the error in less time while the Default and Example types had the exact opposite effect and only cause confusion for the students. This research focused on compiler feedback and did not include code smells. The research did show that the manner of displaying the feedback influences the student's ability to solve the issue.

2.3 Impact of compiler feedback

Abegaz and Spence[9] investigated the relation between affect and problem solving. They recruited sixty-three students to take part in the experiment. Each of the student got a small program with a few lines of code, which was constructed to produce errors. Each of the participant was tasked to compile the program and fix the errors that arose until the program compiled successfully. Each participant was given 9 tasks to complete, receiving customized feedback for each task. The hypothesis was that feedback promoting positive affect would have a measurable impact on the performance. The conclusion however was that the feedback which promoted negative affect were more likely to increase the performance of the participants. This research also focused on compiler feedback and error messages and did not include code smells. The research did show that sentiment in the feedback influences the student's ability to solve the issue.

3. METHODS

The data set used for this study is the data of one course. For this course a prototype is implemented, which will also be used in the future for further research. The data set of the current course is the first data set and more will be added in the future. This allows for a longitudinal observational study of the effectiveness of feedback on code.

3.1 Data set

The data set used in this research is extracted from the Atelier platform. Atelier stores data on users, projects, source files, and feedback (comments) on the code. A project is a bundle of source files needed to create a working program. Projects can be re-uploaded to create new feedback on the project.

The data set extracted from the Atelier program contains the data of an ongoing course with 145 students and TAs. The Atelier tool was rolled out at the beginning of the course. The data set thus may be somewhat incomplete. Users of the Atelier program have to give consent in order to be included in the extraction of the data.

3.2 Data analysis

For the analysis of the data a small prototype was created. This prototype was run on all projects of the data set. The prototype is integrated into the Atelier platform for future use.

3.2.1 Classification of feedback

In order to analyze the data the comments had to be classified into different categories. Comments are classified in two different ways. First they are classified on the type of code smell. These categories are used to group issues together. Second, comments are classified on which type of feedback it is. These categories are based on the way of creation and the level of interaction the comments provide. The used categories are created as result of literature review and interviews with TAs.

3.2.2 Measuring effectiveness

To measure the effectiveness of different types of feedback we first need to know what effective feedback is. To find an definition for effective feedback an literature review was performed accompanied by interviews with TAs.

3.2.3 Implementation

The implementation of the model is done in the *Atelier* project. The data is retrievable via an API call for which sufficient permissions are required. The first step the tool takes is to retrieve all users which has given consent. After retrieving all users, the program start to calculate the metrics from the previous section on a per user base. The program first checks the solving of the issue within a project. For this all projects of the users are checked. The program categorizes the comments of the projects and check which comments are solved in the next version of the project and which remain unsolved. Next the program checks the re-occurrence of the issues in later projects. The program retrieves the unique categories per project and counts how often those categories return in later projects. After calculating this for all users, the data of all users is aggregated resulting in the final numbers.

3.2.4 Limitations of the implementation

There are a few problems with the implementation. The Atelier project has, at the time of writing, no versioning of projects. Users can upload folders to create a submission. These submissions are currently grouped by name to create the projects used in the calculations. The Atelier platform also does not store information about the type of code issue which a comment addresses. For the automatic generated comments this is trivial since most of the messages can be mapped directly to the original issue type. However in the case of the manual comments, the mapping has to be done manually. Some pattern matching is used to categorise most of the comments, the rest of the comments had to be assigned a category one by one. This might lead to some errors in the data when used on a different data set.

4. RESULTS

Table 1 shows some general information about the data set. Comments are not always available for the students to see. When a comment is visible to students it is a 'public' comment. When students can not see a comment, the comment is 'private'.

Table 1: Data Set Information

Unique projects	233
Total number of submissions	305
Public comments	458
Private comments	1232

4.1 Categories of code smells

Stegeman et al.[1] constructed a model for feedback on code quality. In this model they describe nine categories of feedback. For this research we group the headers and comments category together under 'documentation' and we disregards the layout and formatting categories. This results in the use of the following six categories used in this research:

Names Issues relating to naming of classes and variables. Names should be descriptive of what the code does.

Documentation Issues relating to headers and comments. Header comments should briefly summarize the goal of separate parts of the program. Comments throughout the code should give extra explanation about decision and working of the code.

Flow Issues relating to control flow componentes of the code. Control structures should be nested properly. Duplicated and unreachable code should not occur.

Expressions Issues related to expressions that are too complex and data types that are unsuitable.

Decomposition Issues related to length of variables and methods. Methods should be limited in length and perform a limited amount of tasks. The amount of parameters passed to a method should not be too great.

Modularization Issues related to classes without a clear goal. Classes should have a clear purpose and not have too many functions and variables.

The comments generated by *Zita* are all associated with an PMD issue. These issues were investigated and associated with the mentioned categories. Table 2 consists of a list of issues per category. Per issue the total amount of comments and the amount of public commments is shown. The comments given by users are harder to divide into reusable issues, for this reason the comments generated by users were only associated with the aforementioned categories on not with separate issues.

Table 2: Issues per category

	all	public
Names		
ShortMethodName	1	0
AvoidFieldNameMatchingTypeName	10	5
AvoidFieldNameMatchingMethodName	11	2
ShortVariable	276	23
LowerCaseVariables	88	5
LowerCaseMethodName	34	8
Documentation		
UncommentedEmptyConstructor	27	3
UncommentedEmptyMethodBody	34	0
Flow		
StdCyclomaticComplexity	2	1
EmptyStatementNotInLoop	5	2
EmptyIfStmt	1	1
UnusedFormalParameter	16	2
UnusedLocalVariable	4	2
StatelessClassRule	15	1
Expressions		
AvoidReassigningParameters	14	0
AtLeastOneConstructor	6	0
DecentralizedDrawingRule	129	22
DrawingStateChangeRule	157	4
DecentralizedEventHandlingRule	141	22
SimplifyBooleanExpressions	19	4
PixelHardcodeIgnoranceRule	234	19
Decomposition		
LongParameterListRule	8	3
LongMethodRule	5	2
Modularization		
TooManyFields	3	1
GodclassRule	0	0

4.2 Categories of feedback

Next to the the classification on code smell, the comments were also classified based on the creation of the comment and the level of interaction it provides. In the Atelier system comments are automatically generated by the Zita plugin, next to this manual feedback is provided by other users. Users can respond to each others comments and enter into discussions or ask questions for clarification. For this reason the feedback is classified into the following categories:

Automated These comments are automatically generated by the Zita plugin.

Manual These comments are manually given by users of the system. In this research those users are teachers and teaching assistants.

Discussion In this type there is interaction between the students and teachers/teaching assistants. In this type both a student as well as a teacher or teaching assistant need to have responded on a comment

Question These comments include a question. This type is added in the hope that questions trigger the students to better think about their problems.

Table 3: Number of comments per class

	total	public
Automated	1327	121
Manual	262	255
Discussion	88	77
Question	13	5
Total	1690	458

4.3 How to measure effectiveness

Cordova and Smith state that students need to know three things in order to learn: What does good performance on a task mean; how does their performance relate to good performance; and how can this 'gap' be closed.[10] This means that in order to learn, students must know how to increase their performance.

The question to what is effective feedback was also asked to teaching assistants in programming courses at the University of Twente. Their answer support the statement made by Cordova and Smith. In order for feedback to be effective, students must know how to fix the problem and know what was wrong so they will not make the same mistake later.

This can be translated into the following two metrics which are used to measure the effectiveness of feedback on code quality:

Solving the mistake The first aspect is that students know how to solve their mistake. This can be retrieved from the data by checking whether students solve the issues in later versions of the same project.

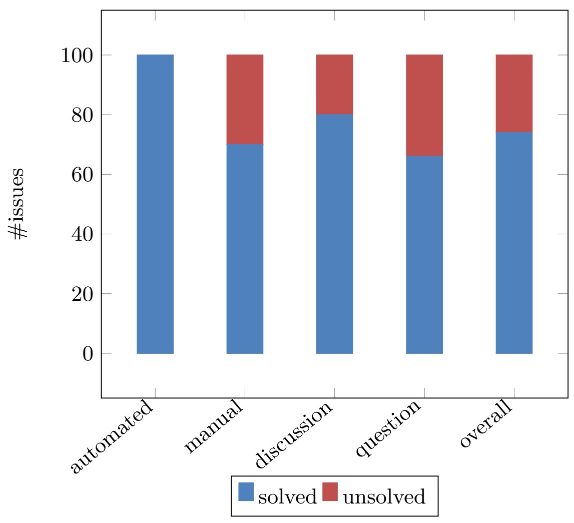
Not repeating the mistake The second aspect is that students do not repeat their mistake. This can easily be retrieved from the data set by checking whether mistakes made by students occur in later projects uploaded by them.

Because of the nature of the metrics some of the data has been excluded from the results. These were the projects of which only one version was ever uploaded, as well as the users which only uploaded one project. The rest of the data was used to generate the results.

4.4 Solving mistakes

The first aspect for the analysis was the metric of how often a mistake is solved. This can be checked by seeing whether or not a later version of a project contains the same issue. Figure 1 shows how often issues are solved for the different types of feedback. The figure shows the the percentage of issues solved for the different types of feedback.

Figure 1: Issues re-occurence

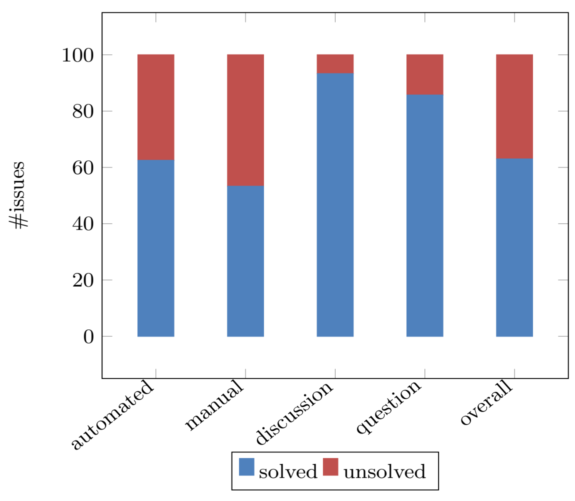


The figure only shows project of which 2 or more versions were uploaded. In these projects only four automated comments were shown to the students. These comments were fixed in all four cases. The manual feedback is by far the most given type of feedback. About 75% of the issues of this type were solved in the next versions of the project. The discussion and question type only appeared a few times in the projects. Not all of them are solved. This can be described by the fact that discussions and questions might end in the teacher/teaching assistant dismissing the issue which leads to it not being solved.

4.5 Re-occurrence of mistakes

Figure 2 contains information about how often issues are solved in later projects after students have received feedback on them. The figure shows the the percentage of issues solved for the different types of feedback. Appendix 1 contains more elaborate figures with a figure showing the re-occurrence of the different categories of code smells for different types of feedback.

Figure 2: Issues re-occurence



This figure only shows data of users which uploaded at least 2 projects. As we can see, overall, issues are re-occur in later projects around 40% of the time after having received feedback on them. The automated and the manual type both have about the same percentage of issues re-occurring. The issues on which feedback of the question and discussion types was given barely return. This might indicate that interaction between students and teachers/teaching assistants might lead to a better understanding of the problem.

4.6 Insights obtained

From figures 1 and 2 we can conclude that issues are more often solved within new version of a project while they return more often in later projects. This might indicate that students know how to fix the problems but do not yet truly understand why it was an issue in the first place. Furthermore we can see that within a project the automated feedback is fixed every time while the other three types have a lower rate of solving. Issue re-occurrence is lower when the feedback is given in discussion or question form. This might indicate that discussions and questions might lead to better understanding of the issue.

4.7 Extension of atelier

The model described in this research is integrated into the atelier platform. The statistics can be requested per type of feedback or for all types at once. If it is requested for a specific type, the extension returns the data per issues category. If the data is requested for all types at once, it only returns the sum of the categories. The data can be viewed on the platform via a dashboard accessible to users with sufficient permissions. These permissions are currently only granted to teachers but can easily be granted to different users.

5. DISCUSSION

One of the findings in this paper is that students often fix their problems after receiving feedback on them. Some types of feedback are more effective than others. In this research the automated feedback has the most impact. Compared to Hieke et al. [7] the percentage of issues fixed is significantly higher. One reason for this might is that this research focuses on issues on which feedback was given. When including issues in the code on which the student did not receive feedback, we can see that the data points starts to match. Another reason might be that this research evaluates projects instead of separate source files.

The data set itself also has some issues. First of all the data set is rather small. There is only data of about 6 weeks with around 20 to 30 students actively using the system. Next to this the Zita plugin was updated halfway through the data collection. This upgrade added lots of new issues which were checked. This results in the first few weeks containing far less issues than the later weeks in the data set. We also do not have data about whether a student has actually seen the provided feedback. Students can upload projects and never open them again.

A last point of discussion is the discussion and question categories for the feedback in relation to the data set. The discussion type is created to check whether active discussions and interaction increases the ability of the students to understand the problems. The atelier platform itself is still quite new and not fully used as intended yet. Other media is often used as contact to ask some questions or discuss some things with the students. This decreases the discussion done in atelier. Both categories have an significantly smaller amount of comments than the other two categories. This makes it difficult to make a fair comparison between the different categories. .tex

6. CONCLUSION AND FUTURE WORK

In this research we explored the measurement of effectiveness of feedback on student's code. We developed four categories of feedback which can be used to classify feedback. We developed a definition for effective feedback and a method to measure the effectiveness of feedback. This method was implemented in a prototype. An initial analysis of the data provided some interesting insights. We found that the automated feedback leads to the most issues being fixed within a project while the discussion and question types both lead to issues re-occurring the least in newer projects. This leads us to believe teachers should engage with students and have discussions to increase the effectiveness of the feedback.

6.1 Future Work

Further research is required to better understand how students deal with the provided feedback. Furthermore, more data is needed in order to solidify the conclusion drawn in this paper.

Further research is also required to check whether affect in the feedback influences the effectiveness of the feedback. Abegaz and Spence already investigated the relation between affect and problem solving. However, their study does not focus on issues regarding code quality.

6.1.1 Future use of the extension

Finally we have some recommendations for further use of the developed extension. The Atelier platform currently requires TAs to make feedback public before students can view the feedback. This leads to a lot of generated feedback not being shown to students. The current data about automated feedback is thus dependent on the TAs. This can be changed so that automated feedback is always shown to students, this was we can truly see the effectiveness of automated comments.

The comments are linked to each other quite rudimentary. The issues of the same type might not always the same issue, however some might be handled as such in the current version. An machine learning algorithm may be developed to better understand what type of issues are present and which issues re-occur in later version/projects.

The extension currently generates some basic statistics which can be requested by the users. However, it might be a good addition to enable users to write their own queries. This can help them draw other conclusions about the data.

7. REFERENCES

[1] E. Barendsen M. Stegeman and S. Smetsers. Towards an empirically validated model for assessment of code quality. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, page 99–108, ACM, 2014.

[2] Martin Fowler. Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.

[3] John Hattie and Helen Timperley. The power of feedback. volume 77, pages 81–112, 2007.

[4] A. H. Mader and E.C. Dertien. Tinkering as method in academic teaching. In *18th International Conference on Engineering and Product Design Education: Design Education: Collaboration and CrossDisciplinarity.*, 2016.

[5] J. Lave. Situating learning in communities of practice. In *Perspectives on Socially Shared Cognition*, page 63–82. American Psychological Association, 1991.

[6] R. de Man and A. Fehnker. The smell of processing. In *Proceedings of the 10th International Conference on Computer Supported Education*, volume 2, page 420–431. INSTICC, SciTePress, 2018.

[7] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, page 110–115, New York, NY, USA, 2017. Association for Computing Machinery.

[8] H. B. M. Kadekar, S. Sohoni, and S. D. Craig. Effects of error messages on students' ability to understand and fix programming errors. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2018.

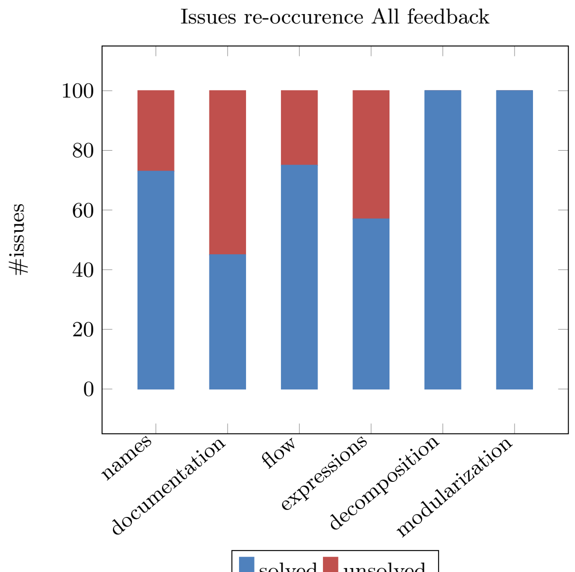
[9] Tamirat T. Abegaz and Dianna J. Spence. Impact of compiler's feedback on coding performance. In Sakae Yamamoto and Hirohiko Mori, editors, *Human Interface and the Management of Information. Visual Information and Knowledge Management*, pages 265–279, Cham, 2019. Springer International Publishing.

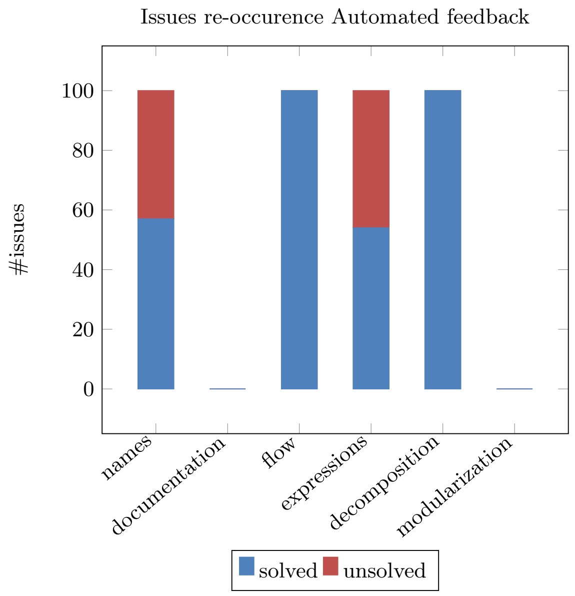
[10] L. Smith and J. Cordova. Weighted primary trait analysis for computer program evaluation. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, volume 20, page 14–19, Evansville, IN, USA, June 2005. Consortium for Computing Sciences in Colleges.

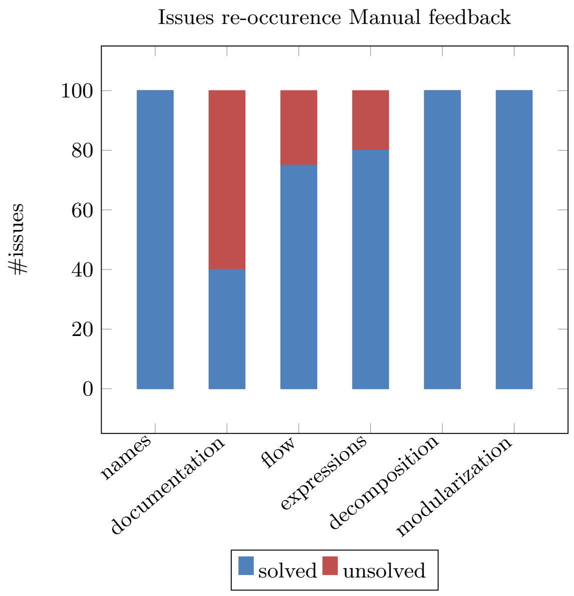
Appendices

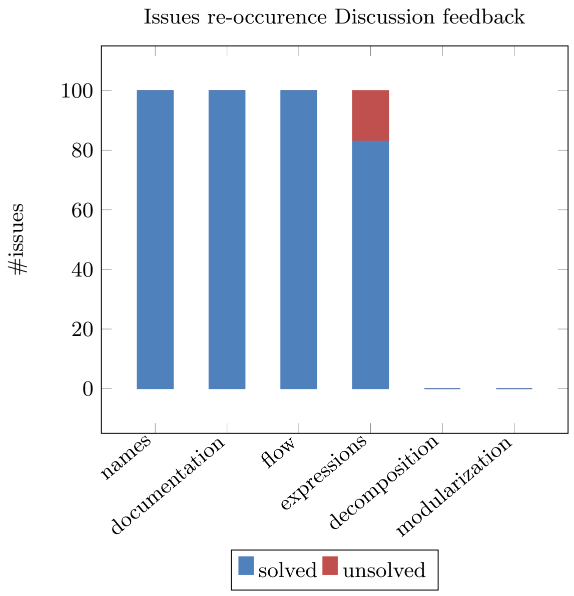
A. APPENDIX 1

Issue occurrence in later projects per issue type per feedback type. Question type is excluded due to the small amount of comments of this type.


Issues re-occurence All feedback
#issues
names
documentation
flow
expressions
decomposition
modularization
solved
unsolved


Issues re-occurence Automated feedback
#issues
names
documentation
flow
expressions
decomposition
modularization
solved
unsolved


Issues re-occurence Manual feedback
#issues
names
documentation
flow
expressions
decomposition
modularization
solved
unsolved


Issues re-occurence Discussion feedback
#issues
names
documentation
flow
expressions
decomposition
modularization
solved
unsolved