Investigating the correlation between prior programming experience and cyclomatic code complexity in student software projects

Collaboration in Computer Science Education

Beatriz Barroso[1]

Supervisors: Dr.ir. E. Fenia Aivaloglou[1]**, MSc Merel Steenbergen**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

Name of the student: Beatriz Barroso[1]
Final project course: CSE3000 Research Project
Thesis committee: Dr.ir. E. Fenia Aivaloglou[1], Ir. Merel Steenbergen[1], Dr.ir. Mitchell Olsthoorn[1]

Abstract

Cyclomatic complexity often serves as a proxy for code quality. While it has been previously studied within education, its relationship with prior programming experience remains unclear. This study analyzed the correlation between different experience metrics, including total years of programming experience, years of experience in industry and in academia, and cyclomatic complexity in student projects. A thematic analysis of task difficulty further contextualized the findings. Results indicated mostly no significant correlations between prior experience and complexity metrics, with only two significant correlations found. While total years of experience and experience in industry showed to be negatively correlated, experience gained in academia showed no consistent findings. The findings suggest that experience alone does not explain complexity scores, indicating that when used as an evaluation method in collaborative projects the results may be influenced by contextual factors such as project settings and task types.

1 Introduction

Collaboration in software development is central in current computer science education practices [1]. Yet, assessing individual student contributions still poses challenges [2]. Version control systems and tools provide useful insights, but overlook contextual metrics such as task difficulty or student's prior programming experience. Therefore, understanding how their previous experience affects structural complexity, such as cyclomatic complexity, could offer insights into individual progress and contribution value.

The influence on prior programming experience on software quality has been extensively researched [3]. Still, a clear relationship is not yet established. Different metrics of experience and quality have been explored, with evidence that higher project experience leads to fewer bugs [3, 4]. Additionally, experience within academia has been found to have a positive effect on code quality and developer productivity [5].

Cyclomatic Complexity is an important metric to assess code quality [6], as it can be an indicator of testability, maintainability and possible vulnerabilities. It is able to assess performance across different languages and programming styles, which makes it very versatile [7]. While it is a widely used metric to evaluate quality [8] and has been studied within educational contexts [9], the effect of programming experience on its performance has not been researched.

As such, the following research questions were formulated:

- **RQ1** How does prior programming experience affect cyclomatic complexity?
- **RQ2** What is the relationship between prior programming experience gained within different contexts (academic vs industry) and the cyclomatic complexity score?

To answer the first research question, both quantitative and qualitative data will be used. The quantitative investigation will compare performance across cyclomatic complexity measurements. Furthermore, patterns between task difficulty and the quantitative results will be evaluated through a thematic analysis of scrum meeting transcriptions. There is an expectation that prior programming experience **negatively** correlates to maximum cyclomatic complexity and **positively** correlates to total cyclomatic complexity. It is expected that, experienced students write code with lower complexity scores and higher efficiency, but that overall contribute in higher quantity.

The second research question focuses specifically on an analysis of experience gained within academia and in industry. Prior research [5] found that experience gained in academia has a positive effect with code quality, whereas industry experience has no relationship. This study will test whether the links hold when considering cyclomatic complexity.

The report is divided in the following sections: Section 2 introduces related literature and background information on cyclomatic complexity; Section 3 introduces the approaches followed to answer each research question; Section 4 describes the results found per research question; Section 5 analyses and considers the different ethical considerations of the study evaluating reproducibility, replicability and threats to validity; Section 6 provides an interpretation and discussion of the results; Section 7 summarizes the main findings, outlining different approaches and research questions for future research.

2 Related Work

The evaluation of student contributions within software engineering projects has been found to be hard to assess. This is due to the complexity of analyzing and discerning individual student contributions, as there are many quantitative and qualitative metrics that can be chosen [1, 2, 10].

Within collaborative student projects, version control systems are often relied upon to determine value of individual contributions and teamwork [2]. Tools such as GitReporter and GitCanary [11, 12], have been developed to integrate with version control systems and aid the teaching teams in assessing contributions. Both of these systems aim to provide further insights on the quality of contribution of the students, analyzing details on code quantity (e.g., lines of code (LOC)) and quality measurements (e.g., cyclomatic complexity) and considering short-comings of current version control systems (e.g., counting documentation as implementation instances) [11, 12]. While these tools are able to assist in grading, it is important to consider their limitations. Guttmann, Karakas, and Helic [11] reported that, as students learn of the metrics evaluated, it is possible that they could change their behavior to attain higher scores. Furthermore, as the tools only consider static code analysis they fail to capture contextual factors such as task difficulty, importance of contributions and student programming background.

During collaboration, it is common for students to have different levels of experience in programming [1], which can be, for instance, due to personal involvement on extracurricular projects, previous education or work experience. Because of this, it can be hard to assess the student's learning experiences. The effect of prior programming experience on code quality has been researched extensively [3, 5, 13]. Lopes, Oliveira, and Figueiredo [3] conducted a systematic literature review in which they mapped the different metrics used in previous research to determine prior programming experience and code quality of contributions, as well as the relationship between these.

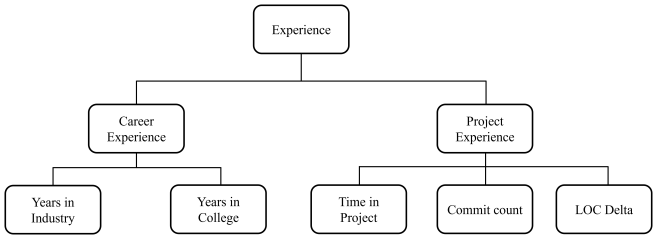

Figure 1: Prior experience measurement categories as outlined by Lopes, Oliveira, and Figueiredo [3]

It was found that prior experience metrics are defined under two main categories: Career Experience and Project experience - Figure 1. Furthermore, code quality was primarily set under Technical Debt and Bugs. The literature review highlights inconsistencies in previous findings, which suggests that the relationship between these metrics might be dependent on context and potentially affected by how experience and quality are measured. These results call for further research, with clearer definitions and separation of metrics, while also considering other aspects of experience such as domain and complexity of projects [3].

The link between Project experience and Bugs was the most studied pattern and the effect of Career Experience has only been studied in regards to Bugs, with only 3 studies having considered it. Two of these studies found a positive correlation of career experience with code quality [14, 15]. The third study found no clear link between the metrics and that the types of programming experience metrics may affect software quality in different ways. Nonetheless, it found that experience gained specifically within academia significantly influences code quality and productivity (positively) [5].

There are different metrics that can be used to evaluate the quality of a program. Previous research has analyzed quality from different lenses [4, 13, 16]. Furthermore, Fenton and Bieman [6] have analyzed and laid out different approaches to evaluate code quality, exploring size and structural integrity measurements. Cyclomatic complexity is indicated as an important tool to evaluate structural integrity, testability and maintainability of the code [6].

Cyclomatic complexity has been analyzed in previous research within educational contexts. Mohamed, Sulaiman, and Endut [9] tried to establish a relationship between student performance in cyclomatic complexity and their respective grade scores during both continuous and final examinations. The study found no consistent relationship between these and suggested a different type of feedback system during the continuous assessment, so that the students could improve their code. Thus, even though cyclomatic complexity has been researched within education, it has not been studied considering larger software engineering projects and student experience.

Although there is extensive research on prior programming experience and code quality, the metrics used are very diverse and few studies have analyzed the impact of prior programming experience on structural quality within collaborative projects. Furthermore, despite version control systems being relied upon in assessment, few have examined this relationship with respect to common Git metrics, namely cyclomatic complexity.

2.1 Cyclomatic Complexity

Cyclomatic Complexity is a mathematical measurement of code complexity which was introduced by McCabe [17]. This metric relies on Graph Theory, considering the different elements of a program as nodes and edges within a graph. It assesses the number of independent paths within it to then determine its complexity score, which is described by equation 1.

$$V(G) = e - n + p \tag{1}$$

Where $V(G)$ represents the cyclomatic complexity score of a graph, G, and e, n and p correspond to its edges, nodes and connected components, respectively. The calculation for cyclomatic complicity also inherently considers nestedness within code as nested elements introduce specific independent paths. This metric is a useful indication of code testability and vulnerability to potential bugs [18]. A higher cyclomatic score implies a higher number of independent paths in the program, which in turn will require a higher number of tests for full coverage. It is a prevalent metric used as a code quality indicator within industry [8] and often introduced in early stages of Software Engineering education.

3 Methodology

3.1 Data Collection

For the course of this project, the research team followed student teams from the CSE2000 - Software Project - course at the TU Delft. In this course, students, as teams of 5, follow a software project for the duration of ten weeks. Each team member from the research team observed a Software Project team, in total 4 teams were observed (N=20).

The data collection procedure consisted of a survey, observance of scrum meetings per team and access to Git repositories. This course uses GitLab as the main development platform, where each student commits and makes changes independently of their peers. Consent for observance was obtained beforehand, via the consent form in Section A.1, and is further described in 5.

Prior programming experience was evaluated through a self-reporting survey, which can be found in A. The survey is composed of different questions from Feigenspan, Kästner, Liebig, Apel, and Hanenberg [19]. These target the type, duration and complexity of each student's experience in programming. Some additional questions were added to capture more specific information about their academic path at the TU Delft and their professional career (questions 8.4, 8.5 and 13 in A.3 were added to the existing survey)

To evaluate **Cyclomatic complexity**, the data was extracted from the respective Gitlab repositories, this process was done using a bash script. The complexities of the code were plotted alongside extraction using Lizard [20]. Out of the four available repositories, two were programmed in languages not supported by Lizard. For one of which, it was possible to adapt the extraction script and successfully analyze the repository. For the later one it was unfortunately not feasible and it had to be discarded as there were no alternative tools to conduct the analysis within the time frame of the Research Project.

In addition, a total of 7 scrum meetings were recorded and transcribed. These were processed locally and labeled, so to map each speaker to each line.

3.2 Metrics

Prior Programming Experience

As outlined in the previous section, a survey was used to determine the prior programming experience of the participants. Table 1, showcases the data features used for correlation measurement.

Table 1: Experience Metrics

Metrics
Total years of experience
Years of experience in industry
Years of experience within the degree
Experience score - aggregated metrics from the survey, each with an equal weight
Self-Reported Code complexity - student perception of own complexity

Following the categorical analysis schema of Lopes, Oliveira, and Figueiredo [3], outlined in 1, the collected metrics have been separated by type. The first metric evaluates career experience, consisting of both years in industry and years in academia. Additionally, this metric captures voluntary experience, such as contributions to open source projects.

The second metric is used to analyze solely years in industry, a subcategory of career experience. Likewise, the third metric is used to capture years in college (academia), the second subcategory of career experience. Furthermore, the forth metric, experience level, consists of metrics corresponding to the Project Experience category (such as LOC Delta) and additional ones that also encompass the type of experience (namely, experience with different types of programming). Lastly, code complexity will be used as a standalone metric to identify student insights on their own complexity performance.

Cyclomatic Complexity Assessment

To measure code quality and capture different aspects of the student contributions two different approaches were taken. The performance of the students was analyzed by issue and by commit. The two different approaches allow for a detailed analysis of student performance and behavior. Commits are one of the smaller units of measurement within Git and, thus, allow for a detailed analysis of student programming patterns. Nonetheless, different students might have different commit habits, namely different sizes of commits, so to capture a different dimension of student contributions, the complexity was also measured by issue completed. In total, four different metrics were collected per student, represented in Table 2

Table 2: Complexity Metrics

Metrics	Unit of evaluation
Total complexity	entire repository
Average complexity per commit	single commits
Average complexity per issue	branch (detailed in 3.3)
Maximum complexity	entire repository

3.3 Data Analysis

RQ1

After gathering the Git repositories, an initial visual inspection identified that most of the issues were resolved within one branch and by single students. Branches that had multiple contributors or solved multiple issues, were marked for manual inspection. If the contributions by multiple students had a high overlap and were hard to extract, the branches were removed from the analysis pool. Nonetheless, most of the branches marked for inspection were successfully extracted as the student contributions were often to different methods.

During the extraction, Lizard [20] was used to calculate the complexity for each snapshot of the repositories. The tool computes the complexity of each method and of each file, and, for every changed file, two snapshots with the complexities before and after the change were stored. This was done both by commit and by issue. To evaluate each student's contributions to an issue, the data was only extracted for analysis once the issue has been marked as finished and its respective branch merged. Thus, only the final solution is considered for analysis. In case students modify each other's code, the last version of each modification was taken into analysis (which was identified manually). This allows for different analysis in case students work on each other's branches and review or modify the code base. For this strategy authorship was determined based on commit signature and Git blame annotations. For the commit analysis, the data was extracted and mapped according to the commit author. For both methods, the before and after versions had their complexities calculated and compared. The difference - $\Delta Complexity$ - was used as the complexity score for the changes. This strategy allows for both negative and positive values, taking into account refactored code and improvements of complexity. In this case, authorship was determined based solely on commit signature.

To aggregate the data and identify the code owner, different scripts were developed: a bash script to extract the data from Git, a python script to process and format this data, and an R script to plot the correlation analysis.

After extraction, a python script was used to compile the results where the $\Delta Complexity$ and author per commit (or merge commit in case of the branches) were stored. The data was grouped by author and, from the commit analysis, the total, maximum and average cyclomatic complexity per commit were plotted. It is important to note that when extracting the data per commit, merge commits were ignored as the added files were regarded with authorship of the student performing the merge. The branch analysis provided the additional average cyclomatic complexity per issue (per student) metric.

The data collected from the survey was then processed and appended. To set the experience score metric, questions 7, 8.1-3, 10 and 11 from the survey A.3 were aggregated, its scores were normalized and averaged. Before evaluating the correlations, an initial inspection was conducted to the data. It was initially expected that the data followed linear and normal distributions, however, histograms, scatter plots and the Shapiro-Wilk test results were observed, and proved otherwise. Most of the metrics observed did not follow a normal distribution and were not linear, therefore, the Spearman's rank correlation strategy was chosen to evaluate the dataset as it captures non-linear patterns. To account for underlying group patterns, the analysis was conducted both by group and for the total.

3.4 Thematic Analysis

Each of the team meetings attended was recorded and transcribed. In total 7 meetings were observed. Each of the transcriptions were systematically annotated using a thematic analysis framework [21]. The different student experiences were analyzed using codes that aim to categorize the difficulty of the issues each student takes on.

Each of the codes was grouped into a theme that represents the level of difficulty each task had. The insights were gathered from both individual and group opinions. The extracted themes are outlined in Table 3.

All of the transcripts and recordings will be shared by the Research Team for this and related research during the course of this Research Project iteration.

RQ2

To answer the second research question, the Git extraction procedure described in the previous section was used. The cyclomatic complexity metrics were considered only with regards to the academic and industry experience - measured by questions 3 and 13 from the questionnaire A.3. The previous tests had verified that the data was not linear nor uniformly distributed, thus Spearman's rank correlation chosen.

Table 3: Thematic analysis: extracted themes and codes

Themes	Codes
High Difficulty	new area for the student
	challenging task
	higher difficulty than expected
	takes/will take a lot of time/work
	task is big
	taking more time than expected
Medium Difficulty	student has moderate experience
	new but seemingly not challenging
	takes some time but doable/ok
	hard task took less time than expected
	easy task took more time than expected
	task is doable
Low Difficulty	area of student expertise
	comfortable for the student
	high confidence
	lower difficulty than expected
	task does not take a lot of time
	task will be fast to implement
	task is small
	task is easy
	not a lot of work
Group progress reflections	planning too optimistic
	planning not realistic enough
	.. was general group setback
	takes an unknown amount of time
Task weights	task weight assigned
	task weight mentioned
	task weight reevaluated

4 Results

This section is divided in two parts, quantitative and qualitative results. The quantitative analysis consists of a correlation analysis between the prior programming experience and complexity metrics. The qualitative part involves a thematic analysis in which recorded scrum meetings were coded into different themes. This evaluation was conducted to understand the level of difficulty experienced by the students within their issues.

4.1 Quantitative analysis

The correlations were plotted using Spearman's Rank Correlation, through heatmaps and full tables. In the heatmaps, the relationships are described through color coding of the correlation values: negative correlations are presented in blue and positive ones in red (the values range from 1.0 to -1.0, with the middle, 0.0, represented as white). For the two methods, results were marked as **significant** (*) if $|p| < 0.05$ and **very significant** (**) if $|p| < 0.01$.

RQ1

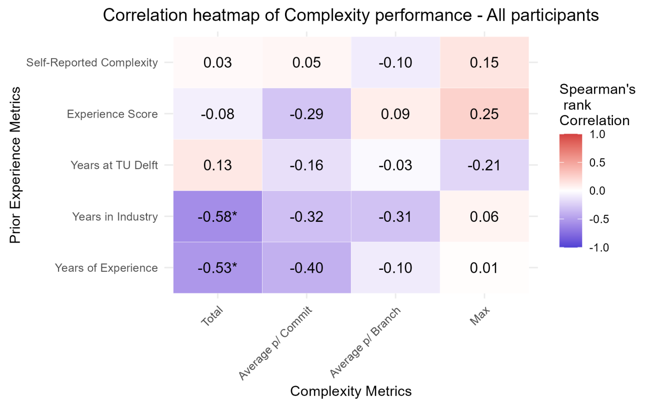

Figure 2: Heatmap of correlations found between prior experience and complexity metrics - All participants

In figure 2, the correlations between all of the metrics have been plotted in a heatmap. These can be observed as well in Table 4 in Appendix B.1, alongside all of the respective p-values. Overall negative correlations were observed between the metrics, with 10 observations having been found belonging to the interval of $[-0.60, -0.10]$. The second most observed trend was of no correlation, with 7 values belonging to the interval $(-0.10, 0.10)$. Some positive correlation scores can be observed, particularly from *Self-Reported Complexity* and *Experience score* towards the *Average Complexity p/ Branch* and *Maximum Complexity*. Two scores were found be significant results ($|p| < 0.05$): the correlations from the *Years of Experience* and *Years in Industry* towards the *Total Complexity*.

To identify group trends the analysis was also conducted for each separate group, which are represented in figures 3, 4 and 5.

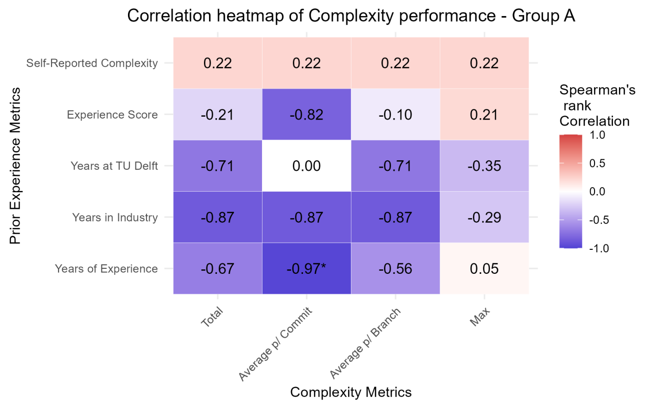

Figure 3: Heatmap of correlations between prior experience and complexity metrics found within Group A

Group A, figure 3, showed to have stronger negative correlations over its metrics, particularly on metrics capturing Time related experience. *Years of Experience* was found to have a strong, significant correlation with the *Average Complexity p/ Commit*. For Group A, the *Self-Reported Complexity* consistently had a weak positive correlation with the complexity metrics.

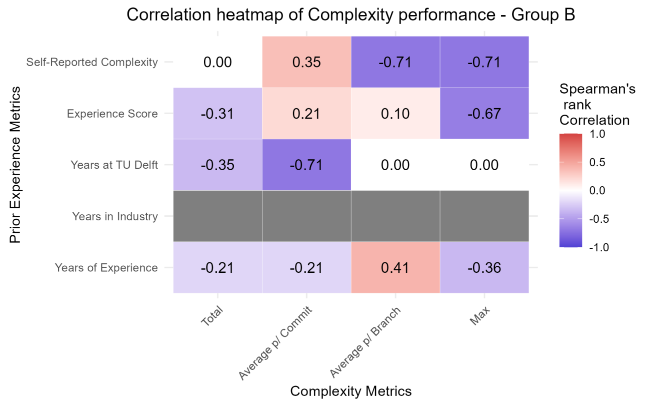

Figure 4: Heatmap of correlations between prior experience and complexity metrics found within Group B

For Group B, most scores showed weaker correlations, both positive and negative. This was the only group where strong negative correlations were observed towards the *Self-Reported Complexity* and *Experience Score* experience metrics.

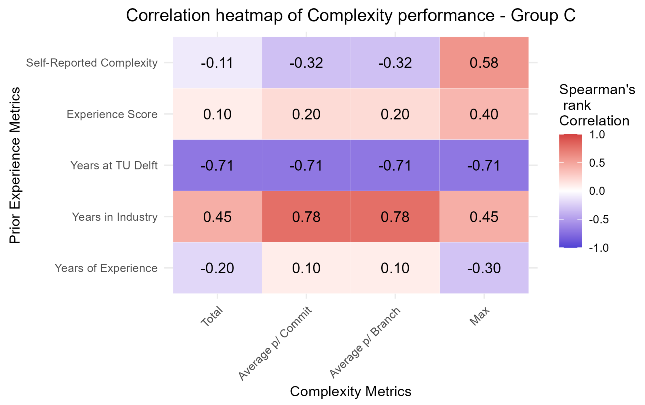

Figure 5: Heatmap of correlations between prior experience and complexity metrics found within Group C

Group C had larger disparities as two metrics showed a positive correlation, one showed a strong negative one and the last metrics showed mixed results. *Years at TU Delft* and *Years in Industry* showed opposing trends, strong negative and strong positive correlations respectively. *Experience Score* was found to have a weak positive correlation towards all Complexity metrics. For groups B and C there were no correlations marked as significant.

RQ2

Figures 6 and 7, portray the correlations found considering the years of experience in industry and years of study at the TU Delft, respectively. For each figure, the correlations have been analyzed both for all of the participants and for each individual group.

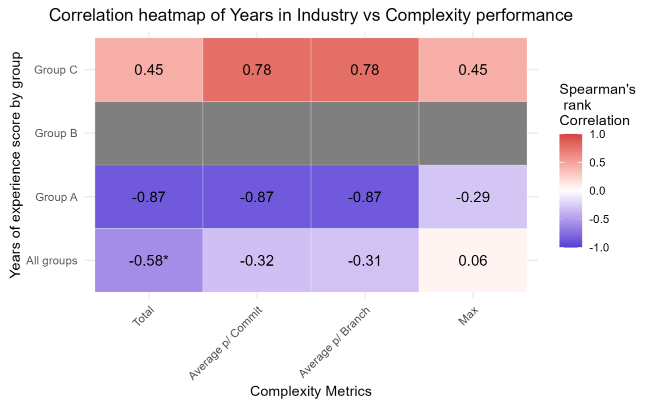

Figure 6: Heatmap of correlations between Years in Industry and the complexity metrics collected (Groups: A, B, C and all).

In figure 6, it is possible to observe different trends per group. While Group A shows strong negative correlations, Group C had an opposing trend, tending towards a positive correlation. Furthermore, Group B did not have any results as there were no participants with experience in industry. When plotting the results for all of the groups, correlations were mostly negative. On a different note, the correlations for the *Average Complexity p/ Commit* and *Average Complexity p/ Branch* consistently had similar scores. Meanwhile, the correlation scores for the *Maximum Complexity* were the closest to 0.0 in every iteration. Lastly, the only correlation marked as significant ($|p| < 0.05$) was for the relationship between the years of Experience of *All groups* and the *Total Complexity*.

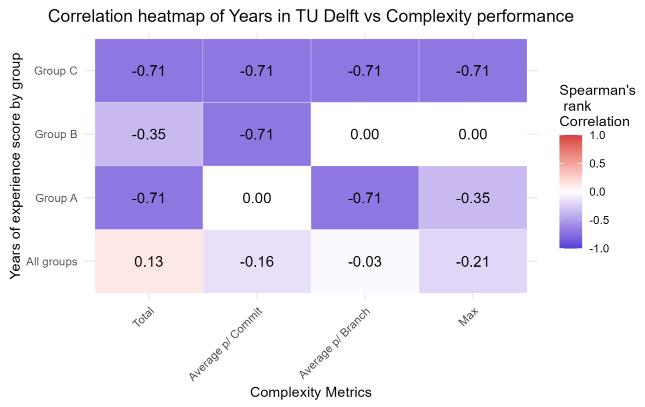

Figure 7: Heatmap of correlations between Years studying at TU Delft and the complexity metrics collected (Groups: A, B, C and all).

The correlation trends for experience gained within academia had multiple records marked as strongly negative. In figure 7, it is possible to observe that there were different patterns in group behavior. For groups A and B there were mixed results: some metrics showed strong negative correlation and others showed either weak or no correlation. For group C, all metrics were found to have a strong negative correlation. When considering all the groups together different trends were observed: *Total Complexity* showed to have a slightly positive correlation score, and the remaining metrics also had weak negative correlations.

The full correlation tables for the two experience metrics have been included in appendix B.2.

4.2 Thematic analysis

In total 7 scrum meetings, from the three different groups, were transcribed and thematically coded. The thematic coding was conducted to evaluate the different levels of task difficulty reported by the students. The goal of the analysis was to assess patterns between the performance of the students considering their prior programming experience as well as the level of difficulty and challenges faced. The thematic coding process followed the thematic schema presented in Table 3. The qualitative analysis was used to answer the first research question.

RQ1

Different group dynamics were observed across the teams. The overall challenges, task distribution practices and perceptions of difficulty varied considerably.

When considering prior programming experience and complexity scores, three major trends were observed:

1. Better performance than expected: low scores on prior experience and good performance on complexity metrics.
2. Performance in line with expectations: low scores on experience with lower scores on complexity; medium/high scores on experience with better scores on complexity.
3. Performance lower than expectations: high scores on experience with low performance on complexity metrics.

The expected trends are defined according to the hypothesis proposed, where a higher level of experience could indicate efficient complexity results.

Most of the students fell within the the second type of pattern, most of them having reported medium to high experience levels. Through the thematic analysis it was found that, generally, these students either take on more tasks or take on tasks with higher difficulty than their peers. Some of which also reported no preference for tasks and were available to take on tasks that other people would not want. Occasionally these students reported challenges when reflecting on the tasks, often describing the issue concretely. An example of a direct quote evaluated is *"I realized my issue was a bit bigger in scope"*.

In total, 4 students had better performance than expected, all of which had good complexity performances, particularly in the *Total Complexity* and *Maximum Total*. When analyzing the task difficulties reported, it was found that these students generally took tasks with all types of difficulty. Nonetheless they tended to report higher difficulty within their tasks. Most of these tasks were either bigger or harder tasks, and often the students described challenges in retrospective, describing specific aspects of the tasks that took longer or how it was more challenging than expected. Some of the students followed the description with requests to increase the assigned task weights. Often the remaining team members confirmed the reflection through quotes such as *"That's completely valid"*. It also happened that other team members reported on the struggles for these students as well.

A number of students was observed in the third category, when analyzing the reports on task difficulty by these students, it was found that, within team meetings, these students had preference for tasks with lower difficulty. These students performed worse, particularly in the *Maximum Complexity* metric, introducing methods with complexity higher than 30. Furthermore, these students were among the ones with highest scores on *Experience Score* and *Years of Experience*. In some cases, the task difficulty reported by these students was not matched with the team's sentiment: while the student felt the task was being harder the team viewed the task as still having low difficulty overall.

It was observed that the teams had different practices within their projects: only some groups explicitly discussed task weights discussed weights; one group considered hours of work but did not take this estimate as weights. Furthermore, some of the groups reported an overall sentiment that their project was of higher difficulty. In some cases, the teams had difficulty making concrete task plans as they were overall not familiar with the project concepts, marking tasks as taking an unknown amount of time. Several groups, revisited their planning in follow-up meetings, evaluating previous estimates: *"We were a bit optimistic at the start"* as an example quote.

5 Responsible Research

This study was approved by the Human Research Ethics Committee (HREC) at TU Delft and was conducted in accordance with the principles of the Dutch Code of Conduct for Research Integrity [22]. Informed consent was obtained from all participants prior to data collections, which was collected by the consent form in A.1 and verbal consent. The dataset was anonymized, with identifiers removed to ensure participant privacy. During the project, all data was stored within TU Delft teams and local servers accessible only to the project team. Upon completion, local data will be erased and the collected transcripts and recordings will be transferred to internal TU Delft servers who will store them for 10 years. To ensure replicability, the anonymized scripts and a detailed description of the methodology will be made available. Nonetheless, reproducibility, will not be feasible due to participant privacy and data protection constraints.

6 Discussion

In this section, the results for each research question have been discussed and evaluated, its applications and limitations outlined. This study explored the correlation between prior programming experience and cyclomatic complexity in student software projects. The findings revealed that total years of experience and years of experience in industry were significantly correlated with lower total complexity, while academic experience showed no consistent relationship. Furthermore, group-level analysis suggested conflicting patterns, possibly explained by Simpson's paradox. These results challenge previous expectations and assumptions on programming experience translates to measurable code quality, through cyclomatic complexity. The section is divided in discussions for each of the research questions, 6.1 and 6.2, Implications 6.3 and Limitations 6.4.

6.1 RQ1

How does prior programming experience affect cyclomatic complexity?

The analysis revealed a negative correlation between prior programming experience metrics -*Years in Industry* and *Years of Experience* - and some cyclomatic complexity metrics, including *Total, Average p/ Commit* and *Average p/ Branch* complexity (Figure 2). Out of these, only two were marked as **statistically significant**: the correlations from these experience metrics towards *Total Complexity*. These findings contradict the initial hypothesis, as students with higher experience had lower complexity sums. This result suggests a trend towards efficiency rather than a higher task volume (with higher total complexity).

As indicated by Lopes, Oliveira, and Figueiredo [3], two studies have linked years of experience with better performance, considering different code metrics [14, 15]. However, cyclomatic complexity has not been studied in this context. Although total cyclomatic complexity was initially considered as an indicator of better performance, our findings suggest that the relationship maybe be more complex, where contextual factors (such as task type or programming paradigms) may play a role.

The thematic analysis added nuance to these findings. Many experienced students took on varied tasks in higher quantity, aligning with expectations. However, students with less experience reported greater difficulty while also maintaining their code quality. Surprisingly, a subset of students with higher experience, tackled simpler tasks yet showed lower performance. These patterns suggest that task difficulty alone does not account for the complexity differences in performance.

Group-based analysis showed additional dynamics. Group A (Figure 3) tended towards strong negative correlations, with a **statistical significant** correlation found between *Years of Experience* and *Average Complexity p/ Commit*. This correlation could suggest that experienced students either made smaller commits, spreading complexity, or inherently wrote simpler code. Other groups showed contrasting patterns (Figures 4 and 5), which hint that experience effects may be context dependent.

Subjective experience metrics, *Self-Reported Complexity* and *Experience Score*, showed contrasting or weak correlations. As the scores were self-reported, the metrics may reflect inaccurate self-assessment. Moreover, the *Experience Score* model, which is based on weighted factors, has not been validated and may have contributed to unreliable results.

It is also possible that collaboration patterns may have influenced complexity metrics. Some experienced students made targetted, smaller commits on teammates branches, often reducing their. This could potentially explain the lack of significant correlations with *Average Complexity p/ Branch*.

In summary, experience - particularly *Years in Industry* and *Years of Experience* - was found to be associated with lower code complexity. However, this relationship varied across groups and task contexts. The findings challenge the initial hypothesis and call for an analysis of student performance considering external context and team-dynamics.

6.2 RQ2

What is the relationship between prior programming experience gained within different contexts (academic vs industry) and the cyclomatic complexity score?

Experience in Industry showed a **statistically significant** correlation with *Total Complexity*. This suggests that students with higher professional experience code with lower overall complexity. As in the previous section, since neither the number nor the type of tasks were measured, it is unclear whether this reflects lower workload or simpler implementations, with less complexity. Furthermore, the different groups showed very distinct results, with one group having moderate to strong positive correlations and while another showed consistently strong negative correlations.

In contrast, *Years at TU Delft* showed no significant results with complexity metrics. While each group displayed negative correlations, when analyzing the full dataset, the pattern was not found. In fact, one correlation became weakly positive instead. This difference suggests that aggregation concealed effects observed within groups, being a possible observation of Simpson's paradox [23], in which trends reverse upon combination of data.

The results do not support the hypothesis that experience gained in academia, measured by Years of study, correlates to better complexity performance. Furthermore, while prior research found that programmer quality and productivity were independent from experience in industry [5], it is unclear whether our results fully support this. As such, the mixed results suggest that project and team conditions can play a role in this association.

6.3 Implications

The findings challenge assumptions that prior programming experience leads to higher code quality. Educators can use these findings to better understand student behavior and adapt assessment criteria. The results showed that cyclomatic complexity metrics can portray considerable disparities in larger software projects. Thus, applying it in smaller, controlled assignments could be more useful - for instance, introductory software testing courses. In collaborative settings, educators could recommend teams of mixed student experience, to balance programming practices. Additionally, cyclomatic complexity may be better suited as formative feedback tool, rather than being included in assessment criterion.

6.4 Limitations

Several limitations should be acknowledged while interpreting these results. Firstly, although cyclomatic complexity is a commonly used indicator for code quality, particularly in Object Oriented contexts, it may not generalize across all programming paradigms used in the student projects. Differences in requirements, size and style may affect complexity consistency.

In addition, Total complexity, average complexity per commit and per branch may be hard to interpret without knowledge on student tasks. Although a higher complexity can be linked to higher participation, it could also be caused by denser, more complex code. Additionally, the different project settings could also have impacted Commit and Branch evaluations, limiting their analysis.

There were also logistical constraints. The project took place over 10 weeks, with data collection starting on week 5. Less teams participated due to recruitment difficulties, and, as one language was not feasible for extraction, one group was excluded (N=15).

In addition, the sample may have been affected by recruitment bias. It is possible that students with higher confidence, more comfortable being recorded, were overrepresented, whilst others opted out due to privacy concerns.

Besides these considerations, the survey design also presents limitations. Some questions added to the survey have not been validated and some of them might not capture the intended information (for instance, years of study at TU Delft can overlook other prior education).

Finally, the small sample size (N=15), particularly for those with professional experience (only 5 participants), limits generalizations. Thus, larger-scale studies are needed to evaluate the relationships observed.

7 Conclusions and Future Work

This research investigated the relationship between prior programming experience and cyclomatic complexity in student projects. Results revealed negative correlations between experience - *Years of Experience* and *Years in Industry* - and *Total Complexity*. Contrary to expectations, experience in academia did not show consistent findings, and group variation highlighted potential contextual factors, namely project type. These findings have implications for Software Engineering education. While cyclomatic complexity remains a strong metric of performance assessment, its interpretation may vary for different projects and student experience levels. As such, it should be applied considering team and project contexts. For future research, could analyse the role of cyclomatic complexity across different programming paradigms and types of tasks performed. This approach may help uncover how cyclomatic complexity patterns vary considering project settings. Additionally, testing whether the behavioral patterns unveiled in the thematic analysis are observed in different conditions would be useful to assess the validity of the findings.

References

[1] Miroslav Tushev, Williams Grant, and Anas Mahmoud. “Using GitHub in large software engineering classes. An exploratory case study”. In: *Computer Science Education* 30.2 (Apr. 2, 2020). Publisher: Routledge_eprint: https://doi.org/10.1080/08993408.2019.1696168, pp. 155–186. ISSN: 0899-3408. DOI: 10 . 1080 / 08993408 . 2019 . 1696168. URL: https : / / doi . org / 10 . 1080 / 08993408 . 2019 . 1696168 (visited on 04/25/2025).

[2] Kevin Buffardi. “Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE ’20. event-place: Portland, OR, USA. New York, NY, USA: Association for Computing Machinery, 2020, pp. 650–656. ISBN: 978-1-4503-6793-6. DOI: 10.1145/3328778.3366948. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3328778.3366948.

[3] Jefferson Lopes, Johnatan Oliveira, and Eduardo Figueiredo. “Evaluating the Impact of Developer Experience on Code Quality: A Systematic Literature Review”. In: May 2024, pp. 166–180. DOI: 10.5753/cibse.2024.28446.

[4] Yilin Qiu, Weiqiang Zhang, Weiqin Zou, Jia Liu, and Qin Liu. “An Empirical Study of Developer Quality”. In: *Reliability and Security - Companion 2015 IEEE International Conference on Software Quality*. Aug. 2015, pp. 202–209. DOI: 10.1109/QRS-C.2015.33. URL: https://ieeexplore.ieee.org/document/7322148 (visited on 06/15/2025).

[5] Óscar Dieste Tubío, Alejandrina Aranda, Fernando Uyaguari, Burak Turhan, Ayse Tosun, Davide Fucci, Markku Oivo, and Natalia Juristo. “Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study”. In: *ICSSP ’18: Proceedings of the 2018 International Conference on Software and System Process*. May 2018, pp. 111–112. ISBN: 978-1-4503-6459-1. DOI: 10.1145/3202710.3203163.

[6] Norman Fenton and James Bieman. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. 3rd. USA: CRC Press, Inc., 2014. ISBN: 1-4398-3822-4.

[7] Dolores R Wallace, Arthur H Watson, and Thomas J McCabe. *Structured testing : a testing methodology using the cyclomatic complexity metric*. en. Tech. rep. NIST SP 500-235. Edition: 0. Gaithersburg, MD: National Institute of Standards and Technology, 1996, NIST SP 500–235. DOI: 10.6028/NIST.SP.500-235. URL: https : / / nvlpubs . nist . gov / nistpubs / Legacy / SP / nistspecialpublication500 - 235 . pdf (visited on 05/07/2025).

[8] Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. “Cyclomatic Complexity”. In: *IEEE Software* 33.6 (Nov. 2016), pp. 27–29. ISSN: 1937-4194. DOI: 10.1109/MS.2016.147. URL: https://ieeexplore.ieee.org/document/7725232 (visited on 06/18/2025).

[9] Noraini Mohamed, Raja Fitriyah Raja Sulaiman, and Wan Rohana Wan Endut. “The Use of Cyclomatic Complexity Metrics in Programming Performance’s Assessment”. In: *Procedia - Social and Behavioral Sciences*. 6th International Conference on University Learning and Teaching (InCULT 2012) 90 (Oct. 2013), pp. 497–503. ISSN: 1877-0428. DOI: 10.1016/j.sbspro.2013.07.119. URL: https://www.sciencedirect.com/science/article/pii/S1877042813020077 (visited on 06/02/2025).

[10] J.H. Hayes, T.C. Lethbridge, and D. Port. “Evaluating individual contribution toward group software engineering projects”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* ISSN: 0270-5257. May 2003, pp. 622–627. DOI: 10.1109/ICSE.2003.1201246. URL: https://ieeexplore.ieee.org/document/1201246 (visited on 06/10/2025).

[11] M. Guttmann, A. Karakas, and D. Helic. “Attribution of Work in Programming Teams with Git Reporter”. In: SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education. Vol. 1. 2024, pp. 436–442. DOI: 10.1145/3626252.3630785. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85189288339&doi=10.1145%2f3626252.3630785&partnerID=40&md5=522e9cb4493f977aa69b78f99205f26d.

[12] J.J. Sandee and E. Aivaloglou. “GitCanary: A Tool for Analyzing Student Contributions in Group Programming Assignments”. In: ACM International Conference Proceeding Series. 2020. DOI: 10.1145/3428029.3428563.

[13] Reem Alfayez, Pooyan Behnamghader, Kamonphop Srisopha, and Barry Boehm. “An exploratory study on the influence of developers in technical debt”. In: *Proceedings of the 2018 International Conference on Technical Debt*. TechDebt ’18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 1–10. ISBN: 978-1-4503-5713-5. DOI: 10.1145/3194164.3194165. URL: https://dl.acm.org/doi/10.1145/3194164.3194165 (visited on 06/02/2025).

[14] Valentina Piantadosi, Simone Scalabrino, Alexander Serebrenik, Nicole Novielli, and Rocco Oliveto. “Do attention and memory explain the performance of software developers?” In: *Empirical Softw. Engg.* 28.5 (Aug. 2023). ISSN: 1382-3256. DOI: 10.1007/s10664-023-10316-9. URL: https://doi.org/10.1007/s10664-023-10316-9 (visited on 06/15/2025).

[15] Zahra Karimi, Ahmad Baraani-Dastjerdi, Nasser Ghasem-Aghaee, and Stefan Wagner. “Links between the personalities, styles and performance in computer programming”. In: *Journal of Systems and Software* 111 (Jan. 2016), pp. 228–241. ISSN: 0164-1212. DOI: 10 . 1016 / j . jss . 2015 . 09 . 011. URL: https : / / www . sciencedirect . com / science / article / pii / S016412121500206X (visited on 06/15/2025).

[16] Christopher Hundhausen, Adam Carter, Phillip Conrad, Ahsun Tariq, and Olusola Adesope. "Evaluating Commit, Issue and Product Quality in Team Software Development Projects". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 108–114. ISBN: 978-1-4503-8062-1. DOI: 10.1145/3408877.3432362. URL: https://dl.acm.org/doi/10.1145/3408877.3432362 (visited on 06/18/2025).

[17] T.J. McCabe. "A Complexity Measure". In: *IEEE Transactions on Software Engineering* SE-2.4 (1976), pp. 308–320. DOI: 10.1109/TSE.1976.233837.

[18] Yahya Tashtoush, Mohammed Al-Maolegi, and Bassam Arkok. "The Correlation among Software Complexity Metrics with Case Study". en. In: *International Journal of Advanced Computer Research* 4.2 (2014).

[19] Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. "Measuring programming experience". In: *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. 2012, pp. 73–82. DOI: 10.1109/ICPC.2012.6240511.

[20] Terry Yin. *terryyin/lizard*. original-date: 2012-06-21T11:31:46Z. May 2025. URL: https://github.com/terryyin/lizard (visited on 05/17/2025).

[21] Virginia Braun, and Victoria Clarke. "Using thematic analysis in psychology". In: *Qualitative Research in Psychology* 3.2 (Jan. 2006). Publisher: Routledge _eprint: https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa, pp. 77–101. ISSN: 1478-0887. DOI: 10.1191/1478088706qp063oa. URL: https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa (visited on 06/02/2025).

[22] KNAW, NFU, NWO, TO2-Federatie, Vereniging Hogescholen, and VSNU. *Nederlandse gedragscode wetenschappelijke integriteit*. en. 2018. DOI: 10.17026/DANS-2CJ-NVWU. URL: https://phys-techsciences.datastations.nl/dataset.xhtml?persistentId=doi:10.17026/dans-2cj-nvwu (visited on 06/02/2025).

[23] E. H. Simpson. "The Interpretation of Interaction in Contingency Tables". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 13.2 (1951). Publisher: [Royal Statistical Society, Oxford University Press], pp. 238–241. ISSN: 0035-9246. URL: https://www.jstor.org/stable/2984065 (visited on 06/22/2025).

A Questionnaire

The questionnaire used to measure prior-programming experience consisted of the sections A.1, A.2 and A.3. The questions in A.3 were extracted from Feigenspan, Kästner, Liebig, Apel, and Hanenberg [19].

A.1 Consent

You are being invited to participate in research about Student Software Development Project Groups. This study is being done as part of the CS3000 Research Project at the TU Delft. The data will be used for analysis by up to five BSc students working on the Research Project and their supervisor. The survey will take approximately 20 minutes to complete. We ask that you fill out this survey honestly, and to the best of your ability. One question asks for the link to your repository, by filling this in, you consent to the researchers having read-only access to your repository for the duration of the Research Project.

As with any online activity the risk of a breach is always possible. To the best of our ability your answers in this study will remain confidential. Participating in this research does not impact your course performance, and the course instructors do not have access to the data. We will minimize any risks by storing the data on TU Delft servers only. We will be collecting your email address on this consent form for consent purposes and for linking your survey answers to the observation data, but this will not be used in data analysis and can only be accessed by the responsible researcher. This form will not be stored together with the survey answers, but in a separately encrypted space to minimise the risk of a data breach. After this research, the data will be stored for at least ten years in a restricted repository for reproduction purposes, in line with the TU Delft Research Data Framework Policy.

Your participation in this study is entirely voluntary and you can withdraw at any time and for any reason. You are free to omit any questions. You can revoke consent at any time and your data will be deleted, you do not have to provide a reason. Please note that we cannot remove results that have been published already. If you have any questions, you can reach the responsible researcher at

1. By filling in your email address below, you declare that

you have read this opening statement and explicitly consent to participate in this research and the processing of your data.

Answer: ________________________________

A.2 Demographic Information

2. What is your age?
Answer: ________________________________

3. How many years of studying CSE at TU Delft have you completed?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5+

4. What is your gender?

Answer: ________________________________

5. Please enter the link to your GitLab repository, so we can request (read-)access

Answer: ________________________________

6. What is your Gitlab name or email address?

Answer: ________________________________

A.3 Programming Experience

7. How do you estimate your programming experience on a scale from 1 to 10, where 1 is very inexperienced and 10 is very experienced?

Answer: ________________________________

Note: For legibility, question 8 will be presented as a list of sub-questions with their respective answer options. In the original survey these were introduced in a table

8. Please answer the following statements.

8.1 How do you estimate your programming experience compared to your classmates?

Very inexperienced		**Neutral**		**Very Experienced**
○	○	○	○	○

8.2 How experienced are you with logical programming? (This refers to programming based on formal logic, where code expresses facts and rules — examples include Prolog, Datalog, or logic-based exercises in courses like AI or Declarative Programming.

Very inexperienced		**Neutral**		**Very Experienced**
○	○	○	○	○

8.3 How experienced are you with object-oriented programming?

Very inexperienced		**Neutral**		**Very Experienced**
○	○	○	○	○

8.4 How complex do you perceive your code to usually be? (1 = very low complexity, 5 = very complex)

Very low complexity		**Neutral**		**Very Complex**
○	○	○	○	○

8.5 How experienced are you with front-end development?

Very inexperienced		**Neutral**		**Very Experienced**
○	○	○	○	○

9. For how many years have you been programming?

Answer: ________________________________

10. How many programming languages do you know at a medium level or higher?

Answer: ____________________

11. How many courses have you taken that required you to implement source code?

Answer: ____________________

12. For how many years have you been programming for larger software projects, e.g., in a company?

Answer: ____________________

13. For how many years have you been programming outside of academic context (as a job)?

Answer: ____________________

14. How long were the professional projects typically?

- ◯ Not applicable
- ◯ <900 Lines of Code
- ◯ 900-40000 Lines of Code
- ◯ >40000 Lines of Code

B Additional plots by Experience level

B.1 Group analysis RQ 1 - Correlation tables

Table 4: Spearman's rank correlation for each of the Prior Experience and Complexity Metric variable pairs

Experience	Complexity	correlation	p-value
Years of Experience	Total	**-0.53***	**0.04***
	Average p/ Commit	-0.40	0.14
	Average p/ Branch	-0.10	0.72
	Max	0.01	0.96
Years in Industry	Total	**-0.58***	**0.02***
	Average p/ Commit	-0.32	0.24
	Average p/ Branch	-0.31	0.27
	Max	0.06	0.82
Years at TUDelft	Total	0.13	0.65
	Average p/ Commit	-0.16	0.58
	Average p/ Branch	-0.03	0.91
	Max	-0.21	0.46
Experience Score	Total	-0.08	0.78
	Average p/ Commit	-0.29	0.29
	Average p/ Branch	0.09	0.74
	Max	0.25	0.37
Self-Reported Complexity	Total	0.03	0.91
	Average p/ Commit	0.05	0.87
	Average p/ Branch	-0.10	0.73
	Max	0.15	0.61

B.2 Group analysis RQ 2 - Correlation tables

Table 5: Spearman's rank correlation between Years of experience in industry and Complexity Metrics, calculated for each group

	Group	Complexity	correlation	p-value
Years of Experience in industry	**All groups**	Total	**-0.58***	**0.02***
		Average p/ Commit	-0.32	0.24
		Average p/ Branch	-0.31	0.27
		Max	0.06	0.82
	Group A	Total	-0.87	0.06
		Average p/ Commit	-0.87	0.06
		Average p/ Branch	-0.87	0.06
		Max	-0.29	0.64
	Group B	Total	*NA*	*NA*
		Average p/ Commit	*NA*	*NA*
		Average p/ Branch	*NA*	*NA*
		Max	*NA*	*NA*
	Group C	Total	0.45	0.45
		Average p/ Commit	0.78	0.12
		Average p/ Branch	0.78	0.12
		Max	0.45	0.45

Table 6: Spearman's rank correlation between Years of study at TU Delft and complexity metrics, calculated for each group

	Group	Complexity	correlation	p-value
Years of study at TU Delft	**All groups**	Total	0.13	0.65
		Average p/ Commit	-0.16	0.58
		Average p/ Branch	-0.03	0.91
		Max	-0.21	0.46
	Group A	Total	-0.71	0.18
		Average p/ Commit	0.00	1.00
		Average p/ Branch	-0.71	0.18
		Max	-0.35	0.56
	Group B	Total	-0.35	0.56
		Average p/ Commit	-0.71	0.18
		Average p/ Branch	0.00	1.00
		Max	0.00	1.00
	Group C	Total	-0.71	0.18
		Average p/ Commit	-0.71	0.18
		Average p/ Branch	-0.71	0.18
		Max	-0.71	0.18