Code readability: Code comments OR self-documenting code

-

How does the choice affect the readability of the code?

David Tollemark
Sebastian Nielsen

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Contact Information:
Author(s):
Sebastian Nielsen
E-mail: seni12@student.bth.se
sebbe605@gmail.com

David Tollemark
E-mail: datc13@student.bth.se
david.tollemark@gmail.com

University advisor:
Kennet Henningsson
DIPT Institutionen, Department of Software Engineering

Faculty of Computing	Internet	: www.bth.se
Blekinge Institute of Technology	Phone	: +46 455 38 50 00
SE-371 79 Karlskrona, Sweden	Fax	: +46 455 38 50 57

Abstract

Context: Code readability is something every software developer tackles every day. In order for efficient maintainability and learning of a program the documentation needs to be of high quality.
Objectives: This thesis is attempting to show what the general perspective the software developers have. We investigate which method that is preferred and why. We will also look at readability and what similarities and differences there are among students and IT professionals.
Conclusion: We have collected data from several sources. Firstly from a literature review where different papers within the field are presented. Secondly from two separate surveys conducted on students and IT professionals.
From our results we found that documentation is something that software developers heavily rely on and that the need for extensive documentation differs with working experience.

Keywords: Readability, Quality, Comments, Documentation, processes.

Table of Contents

Chapter 1

Introduction

Code readability is the ease with which a software developer or other professionals within the field can understand a written piece of code. A code with high readability is structured in such a way that the recipient can understand the code without being familiar with the system or language.

The readability of a particularly piece of code depends on the context in which it is in. For example, the complexity, syntax, overall structure and graphic[1].

According to Diomidis Spinellis research on Code Documentation. 307 projects on FreeDSB were shown to have less than one comment per 100 lines[2]. As IT professionals, the view of how code should be documented is widely dispersed. But what is the best practice and how do we achieve it? This will be examined to find out what role code readability plays and how small or large changes to the code can increase the readability. The experiment will include two different forms of readability, code comments and self-documenting code. The participants will read selected code for each of the selected readability forms and then answer questions about them. The result will be based on time and correctness of the questions. In addition to the experiment a literature review will be conducted in order to answer some of the research questions fully and strengthen others.

The result will be validated in two ways. Firstly, a pre-experiment will be conducted on students at the department of software engineering. The pre-experiment is conducted in order to get a preliminary study that can be used to improve the final experiment. Secondly, the experiment will be conducted a second time after receiving feedback from the pre-experiment. This will strengthen the design of the experiment thus giving more credibility to the research. Furthermore, the final experiment will be conducted on IT professionals that have experience in the field.

Chapter 2

Background

2.1 Code Comments

Comments in code are the main source of documentation and does not count as compilable code by the compiler. Therefore the main key for understandability, readability and maintainability of the code manifests around code comments.

Comments in code are often considered to be crucial among software developers in order to help them understand the functionality and any quality of the system. Even though well documented code is valued high, it is often neglected or downpriortised[3]. In recent times however, agile methods have started to redefine the importance of documentation since it is only viable as long as it helps to reach a goal. With agile development several crucial questions arises: “How much is enough?” and “what should the ratio of documentation be?”. According to the study "A study of the documentation essential to software maintenance" the documentation is essential to software maintenance. The study showed that code comments are the most important artifact for understanding a system[4].

2.1.1 Group/Class Comments, Block Comments

Group/Class comments is a way of documentation that is often used just once in a class or for specific functions. It often documents expected statements such as input and return values. Additionally, it can also be used to document in a more human friendly flow.

An example of block comments could be:

```
/*
* This is a common way to comment code in
* programming languages such as C++.
*/
```

2.1.2 No Comments

The second way to comment code that this paper will touch on is the "no comments" way of documentation. In agile and extreme programming (XP)[22] development, this is a style of documentation that is more common than in other development methodologies[4].

By choosing not to comment code the process of writing the code would be faster. However, the readability, quality and maintainability has shown to decrease according to research done in the field[4].

2.1.3 Self Documenting

The third and final way of documenting code that this paper will cover is the self documenting style of documentation. In a way this style of documentation can seem like a dream for software developers and are embraced by enthusiasts of extreme programming (XP). XP aims to get rid of “unnecessary” documentation in order to give way of “real work”.

In the essay “Comments Are More Important Than Code” Jef Raskin writes “When programmers speak of self-documenting code, they mean that you should use techniques such as clear and understandable variable names. Instead of n or count, it is better to use a readable, self-explanatory name such as numberOfApricotsPickedToDate. This is a minimalist’s documentation. Nonetheless, it helps—the use of explanatory names, whether of variables, modules, objects, or programs, should be encouraged.”[5]. Could this way of documentation be both effective and readable?

2.2. Related Work

The literature study uncovered several interesting papers on the subject that has analysed how readability and/or quality were affected by different types of documentation. In this part of the thesis seven papers will be presented and summarised.

“A Study of the Documentation Essential to Software Maintenance”[4] analysed what software documentation methods are the most useful in relation to software maintainability and quality. The result of the survey presented in the paper showed that the most important artifacts are the source code and the artifacts that it contains. Data models, which is something that is not covered in this thesis, were also shown to be important first mostly the logical data model[4]. Non technical users valued material typically done during the pre-study time such as requirement, use cases, acceptance testing plan or user manuals. The survey also indicated that general views of the system such as architectural models, vision documents were less helpful.

“Comments are more important than code”[5] analysed that self-documenting code is a good standard for a fast “automatic documentation”. But it’s not completely reliable in all cases. Therefore a standard documentation is required where it’s needed.
Software development methodology such as Extreme programming (XP) encourage documentation types like self-documenting code to get rid of unnecessary documentation.

“Quality Analysis of Source Code Comments”[3] analyses the length of comments and how relevant the comments are. The result of the survey conducted on 16 participants showed that the length of comments does play a role. Inline comments with over 30 words were higher valued than comments that had two words. It was also established that longer comments had a much greater scope than those with less words.

"Enriching Documents with Examples: A Corpus Mining Approach"[6] analysed to what degree software developers used the web to find API documentation and what quality the documentation had. The research found that documentation and code examples were hard to find due to the structure that API:s usually are stored. As an example they did an manual inspection of JDK 5 documents including more than 27,000 methods where only 500 of them (approximately 2%) were explained using code example.

“// TODO: Help students improve commenting practices”[7] analysed that from a learning perspective student have a hard time to create good comments and documentation. The reason why is that students often believe that the code is more important than explaining it.

“CloCom: Mining Existing Source Code for Automatic Comment Generation”[8] concluded that a lack of code comments in the software industry is a common issue. They tested autogenerated code comments on 1005 open source projects to add up new comments there it’s needed. But the study showed that only 23% of the autogenerated comments were good.

“Beyond Generated Software Documentation – A Web 2.0 Perspective”[9] concluded that software that is documented well is easier to maintain and reuse. They tested an Eclipse framework for Web 2.0 applications to increase effectiveness of software documentation.

The above presented work suggests that the field presented in this thesis is of great interest within the field of software engineering. The conclusion of many of the presented papers is that documentation plays a big role in the final cost of the software. Poorly documented software will have a higher maintenance cost due to the poor documentation conducted during the development phase. This directly correlates to a gain in cost in the late maintenance stage where most of the project time and cost is spent in a long life software cycle.

Chapter 3

Method and Design

3.1 Research Questions

1. Why is it important with good code documentation?
2. What makes code readable?
3. What is the most common way of documenting the code?
4. What can be done to increase readability of the code?
5. What differences is there between students and IT-professionals when it comes to code readability?

These research questions focus on which is the most efficient way to write readable code and how to increase the code readability. There are other thesis that are focusing on one specific type of code comment type such as group comment, while this thesis focusing on the three most common ways of documenting code and compare them.

3.2 Research Design

The literature study information was mainly gathered from research databases. All data was from academic journals and had been checked by origin and institution for quality assurance. The research databases used were:

- IEEE Xplore
- Google Scholar
- BTH Summon
- ACM Digital Library

Keywords used for finding information were:

- “Software documentation”
- “Code readability”
- “Code comments”
- “Code documentation”
- “Self documenting code”

These keywords gave us a moderate volume of results. The results were read through in order to sort out the ones that we needed. All literature that we thought was relevant was saved until we started to write this thesis.

We thought that the best way to answer most of the research questions was to make a survey.
To gather a good scope of high knowledgeable participants for the survey, we asked a software development company to assist us with participants.
The decision to gather fewer high knowledgeable participants instead of an online survey that would get us a lot more participants was done because we needed experienced software developers to get good results. We prioritized knowledgeable participants over a large amount of participants in this case.
To answer some of the research questions we gathered select students at BTH to participate in the survey. By doing so, parallels could be drawn between students and the working professionals.
We answered the survey questions by analyzing the data based on correctness of the questions, the participants opinions and the time spent on each question.
We chose to have freetext as answers instead of possibles answers. This decision was made because the data would be more accurate and the answers would be of less weight otherwise. With the free text option there is obvious risks involved such as interpretation problems. On the other hand, multiple choice questions have the risk of leading the participants to an obvious answer or simply making the incorrect answer options too easy to eliminate. That is, some of the options would be obviously wrong while the others would make more sense. To counteract this we asked the user to answer the questions with their own words. This also ensured that the survey was not biased or leading in the possible answers.
The same questions were asked for every code example for a more fair comparison between the comment answers.

3.3 Survey Questions

The survey started out with some pre-questions. These questions were used to get an overview of the knowledge of the participant and what he/she thought about documentation. These questions helped us with some data for research questions 2 and 3. The participants were also informed that the survey were time dependent in order to ensure that the survey would be done in one sitting.

Q1 Pre questions

1. Please specify how many years you have been working in the field of software development.
   <Text>
2. Please enter those programming languages that you have knowledge about and can use.
   <Text>

3. How do you comment your code and what do you consider to be a good standard of documentation?

   <Text>

The next part of the survey started out with a group/class commented code example followed by three questions about the comments. These questions helped us with some data for research questions 4 and 5. They were also time tracked.

Q2 Class/Group comment

Code example refers to appendix 1A Code, Group/Class comments

1. Specify briefly what the code does.

   <Text>
2. How would you rate (1-5) the readability of the code?

   <Rate 1-5, there 1 is very bad and 5 is very good>
3. Add comment (not required).

   <Text>

Two follow up questions about the group/class commented code example (not timed tracked).
These questions helped us with some data for research questions 2 and 4.

Q2 Cont

1. Do you think any additional comments or specifications regarding the code is needed to increase the readability?

   <Yes or No>
2. Add comment (not required).

   <Text>

Next code example was “No commented code”. These questions were time tracked.

Q3 No comments

Code example refers to appendix 1B Code, No Comments

1. Specify briefly what the code does.

   <Text>
2. How would you rate (1-5) the readability of the code?

   <Rate 1-5, there 1 is very bad and 5 is very good>
3. Add comment (not required).

   <Text>

Two follow up questions about the “No commented code” example (not timed tracked).

Q3 Cont

1. Do you think any additional comments or specifications regarding the code is needed to increase the readability?

   <Yes or No>
2. Add comment (not required).

   <Text>

Next code example was self-commented code. These questions were time tracked.

Q4 Self-commented

Code example refers to appendix 1C Code, Self-Comments

1. Specify briefly what the code does.

   <Text>
2. How would you rate (1-5) the readability of the code?

   <Rate 1-5, there 1 is very bad and 5 is very good>
3. Add comment (not required).

   <Text>

Two follow up questions about the self-commented code example (not timed tracked).

Q4 Cont

1. Do you think any additional comments or specifications regarding the code is needed to increase the readability?

   <Yes or No>
2. Add comment (not required).

   <Text>

The survey ends with some final questions about which style the participant prefered.

Q5 Final questions

1. Which of the three alternatives shown in this survey do you prefer?

   <Group/Class Comments, No comments or Self documenting>
2. Which of the three alternatives shown in this survey do you think takes the longest time to write?

   <Group/Class Comments, No comments or Self documenting>

3.4 Hosting and distribution

The survey was distributed as a webpage built in HTML and PHP, it was hosted by one of our computers and was regularly checked so that it was online and the participants could use it.

The survey was designed to be easy and with a quick GUI so that no unnecessary delay for the participants would occur.

Third year students in the Software Engineering program were invited to partake in the survey. They were expected to have good knowledge of programming from the university courses that they had been taking for the past three years. They were also expected to have some working experience. In order to validate the results from the students it professionals that worked with development were also asked to partake in the survey. Firstly the it professionals were expected to have good knowledge and several years within the field. Secondly, the nature of their employment should potentially allow for a higher level of objectivity than other developers hired at one specific company. This is however conjecture and nothing that we will prove in this thesis.

3.5 Analysis of survey

The survey was designed to be anonymous and was conducted in two phases. This was done in order to get a more reliable result since the survey included both students and IT professionals.

First the knowledge base of the participants were established. The participants were asked to state their experience with different programming languages, their work experience and how they typically document code. This was important since more experience typically correlates to a higher understanding of the field. Secondly the participants were asked a series of questions about three different ways of commenting code (See appendix 1A, 1B, 1C for code snippets). The code snippets were selected from open source projects found on github[17] in order to ensure that the selected code snippets were not angled towards a specific style of commenting, see the source for the code in the appendix[18, 19, 20, 21]. The time was logged once a participant entered the page containing the code and logged once more when the user pressed next. This allowed for additional data to be collected which was used to show the correlation between time spent looking at the code and to what degree the user understood the code. Thirdly the participant were asked which of the three alternatives of documenting code they preferred and which, according to them, would take the most time writing.

All data collected during the survey were stored in a MySQL[12] database and separate databases were used for the students and It professionals. The data was then exported in a CSV format to Google Sheets where spreadsheets and graphs were made.

Chapter 4

Literature review

4.1 Importance of documentation?

The literature review found that developers in most, if not all cases prefer to read code that have many comments that are of good quality[3]. This is also backed up by our survey. The participants preferred documented code (Appendix B. Q4 Part 2). The research also found that developers are not too keen on writing the documentation themselves.

When looking at larger software projects and its documentation the result, from the background research in this thesis, indicates that bad documentation is not a result of bad documentation processes, but rather of developers not adhering to specified processes within the company[10]. One of the most common mistakes in documentation is that developers do not want to document code that is not finished and tested. Many times the code needs to be rewritten and thus also the documentation. The timespan until the code is completed and fully tested is often long. The developer is now faced with documenting code that is not newly written. This can especially be problematic if much time has passed between writing the code and its documentation. Time will then be spent on understanding the code once again in order to document it properly which is not cost effective.

A survey of industrial software projects in 2002 showed that satisfaction with documentation is low in terms of understandability[11]. Key practices are often neglected and the organization fails to asses the process with a relevant model.

Research has also found that software developers spend most of their time trying to understand code[13]. This has a direct correlation to the maintenance state of a project where typically 70-90% of the total lifecycle budget is spent. When maintaining and foremostly modifying code the documentation is crucial to help developers understand and modify the code[15, 16]. Other research papers have found that users typically do not read the documentation, instead they usually go by the trial and error method of finding solutions[14]. This could also be applied to software developers trying to understand software that is already written. Typically, the first step is trying to solve the problem without any documentation solely based on experience. First after this step, the user reads the documentation. It is therefore of great importance that the documentation is there and of high quality.

4.2 Summary of the literature study results

Patterns can be found while researching the importance of documentation. Firstly it has been made certain that developers rely heavily on documentation, both that the documentation is present and that it is of high quality. However it has also been made clear that developers typically do not like to document their code. The cause of this could be the underlying structure with revisions and testing of the written code or from the fact that it is time consuming. This can not be fully determined in this thesis, but only interpreted as a likely cause of the problem.

Based on our own findings from the survey, developers typically have the attitude that code should not need extra comments that need extra maintenance. Students on the other hand have a more positive attitude to comments directly in code.

Chapter 5

Survey

5.1 Results

The survey was conducted in two separate parts. The first survey was conducted on a total of 15 students whereof seven completed the survey. The second survey was conducted on a total of 12 IT professionals whereof 10 were consultants and fully completed the survey.

In appendix 2A all data from the survey conducted on students can be found as a graphical representation of how the participants answered each question.

5.2 Results student survey

The bar chart below shows the languages that the participants knew about and could use.

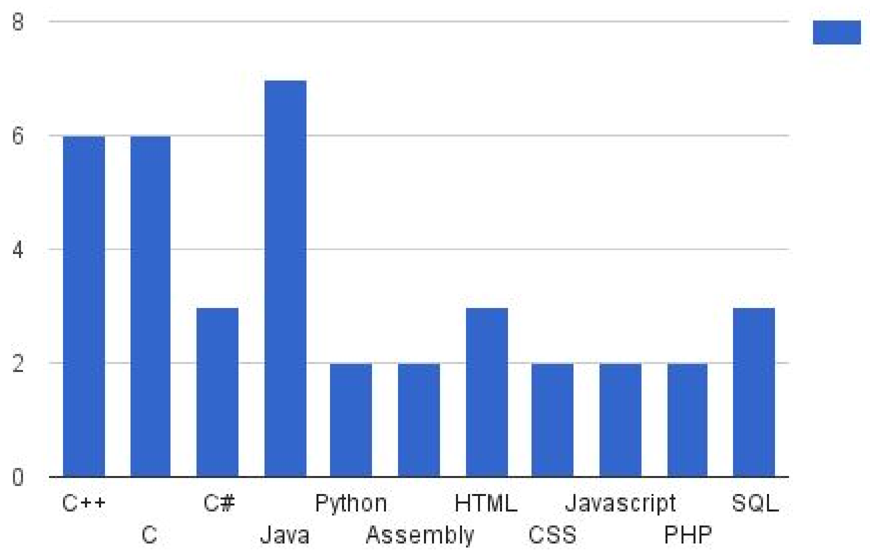


Figure 4.1.1: Graph showing the knowledge base about different programming languages among students.

The pie chart below shows the average time the participants took while looking at the code and answering the questions. For each code example each student spent an average time of:

- Question 1, Group/Class Comments
    - 4.4 minutes
- Question 2, No Comments
    - 3.2 minutes
- Question 3, Self documenting
    - 3.5 minutes

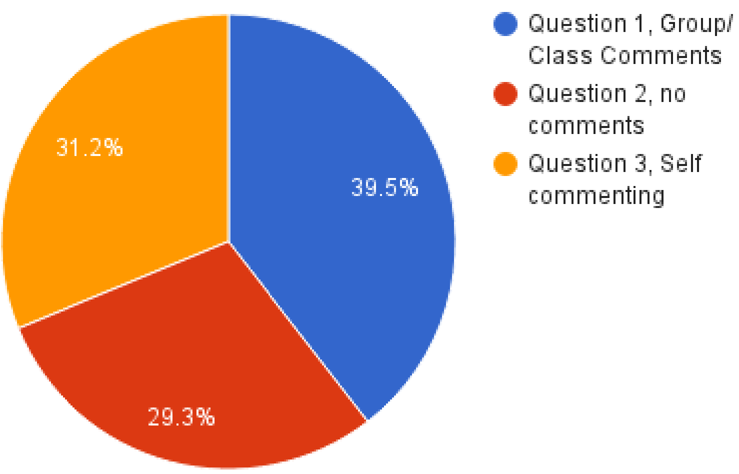


Figure 4.1.2: How long the students took looking at the code and answering the three questions that were logged.

5.3 Results IT professionals survey

The bar chart below shows the languages that the participants knew about and could use. There is a clear analogy between the students and the IT professionals in terms of language skills.

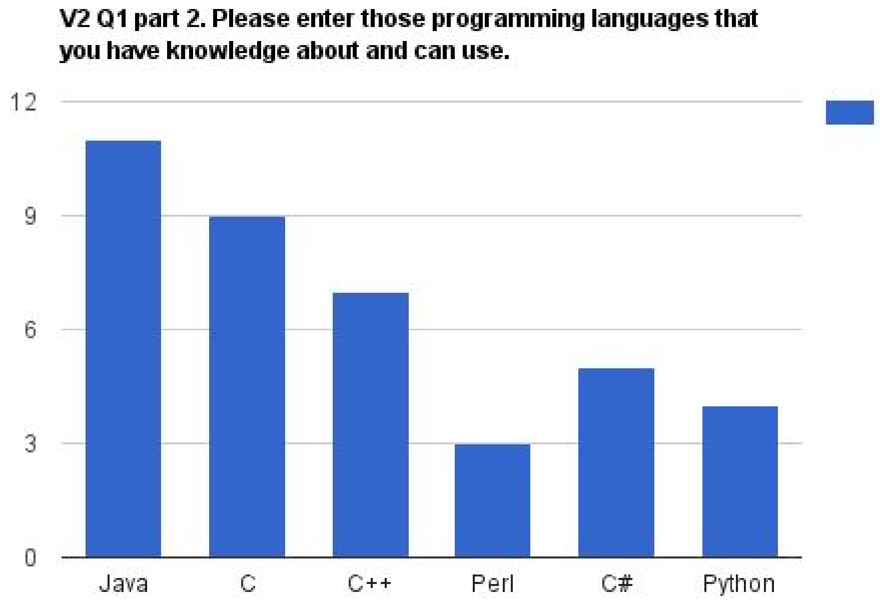


Figure 4.2.1: Graph showing the knowledge base about different programming languages among IT professionals.

The pie chart below shows the average time the participants took while looking at the code and answering the questions. For each code example each IT professional spent an average time of:

- Question 1, Group/Class Comments
    - 3.9 minutes
- Question 2, No Comments
    - 4.1 minutes
- Question 3, Self documenting
    - 4.3 minutes

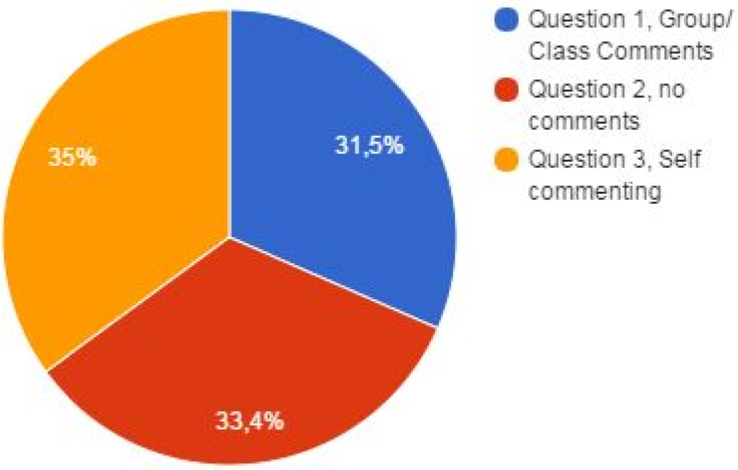


Figure 4.2.2: How long the IT professionals took looking at the code and answering the three questions that were logged.

Chapter 6

Analysis

First, we would like to refer to the appendix part B for the full results and data. All claims stated in this part are based on the collected data. Students and IT professionals are considered to be two separate groups but are also displayed together in this part to show a stronger correlation.

Pre Questions

In order to establish a baseline of the participants knowledge and work experience a series of questions were asked. It is worth noting that the students in all cases answered that their work experience overall were high. This we believe to actually be the years that they have programmed and not the year as professional software developers. On average the students had 3.0 years of experience while the IT professionals had 4.5 years of experience. On average they have 3.9 years of experience in the field of software engineering taken as a whole. The programming languages that most participants had knowledge about and could use were Java, C++ and C with 18, 13 and 15 scores. The language knowledge base correlated well with the languages selected in the survey.

When it comes to the participant's usual way of documenting code and what they consider to be a good standard of documentation, the answers differ from students to IT professionals. Students were shown to prefer Javadocs while IT professional's preferred single line comments. Why the results differ cannot be established in this thesis. However, one could speculate that the students have less knowledge of programming and might require more extensive documentation in order to understand the functionality of the code fully.

Group/Class comments

The three different code snippets displayed in the survey got an overall good score in terms of the participants description of the functionality with 81.25% describing the functionality correctly over all three snippets.

Group/Class comments were interpreted incorrectly by 16.7% of students and 0% of IT professionals. IT professionals rated the readability of the code as very good with a score of 60% while most of the students rated it to be OK at 50%. The average value of both participant groups were 3,4 on a scale of 1-5 (very bad, bad, ok, good, very good). This is considered to be a more than good grade. Whilst the readability rate and the understandability rate were high all students stated that additional comments were needed to increase the readability while only three out of 10 IT professionals stated that more comments were needed.

No comments

The no comments style of documentations also proved to be easy to understand with 0% incorrect from the students and 10% incorrect from the IT professionals. Once again the students stated that the code was either very bad, bad or good with an even score on all three. IT professionals rated the readability as OK and very bad with a majority on 50% on OK followed by very bad at 30%. Students would in four out of seven cases like more comments to increase the readability whilst four out of ten IT professionals agreed.

Self-documenting

The final style of documentation presented in the survey were self documenting code. The students and the IT professionals were given two separate code snippets since the first snippet were considered to be too difficult. Among the students only 50% understood the code and 66.7% stated that the readability was very bad. All students agreed that more comments were needed in order to increase the readability of the code. Among the IT professionals, with a different code snippet, 60% understood the code while 30.8% rated it to be very bad and 30.8% rated it to be good. The difference in score may relate to the participants knowledge about that specific programming language. Eight of the IT professionals stated that more comments were needed to increase the readability while only two stated that no more comments where needed.

Final questions

Both students and IT professionals agreed that group/class comments were the prefered alternative out of the three types of documentation displayed in the survey, 75% of the students and 70% of the IT professionals. Looking at how long the participants thought that the different documentation styles would take to write the students stated that group/class comments and self documenting would take the longest time to write with a score of 37.5% each. The IT professionals however thought that group/class comments would take the longest time to write with a score of 90.9%.

Analyze research questions

In this part we will conclude that we have answered all research questions specified under the method and design part. In order to give credibility to the research we will discuss potential risk and further work in chapter eight and nine. Out of our five research questions the first one is answered under the literature review. The following four questions will be answered below:

R1 . Why is it important with good code documentation?
Answered under Literature review, chapter 4.

R2 - What makes code readable?
In order to determine what makes code readable we needed to compare different types of documentation commonly used. In this thesis we only covered three ways that we determined to be commonly used. Based on the survey we see strong correlations between the software engineers experience and how they would like their code documented. A student might prefer more documentation in order to understand the code while an IT professional tends to lean more towards the less is more model. From the literature review part we also found that in order for the documentation to be useful it should also be easy to find. In order to determine fully what makes code readable more research needs to be conducted.

R3 - What is the most common way of documenting the code?
Based on our survey, the students tend to prefer Javadocs and group/class comments while IT professionals prefer single line comments and self commenting style of documenting. From our textbox responses the IT professionals stated that they used external ways of documentation and almost only commented in the code if the complexity was severe. The same comparison cannot be made among the students. They almost always comment directly in the code where they find the complexity to be great or in areas where they feel that it is needed for other reasons.

R4 - What can be done to increase readability of the code?
In order to increase the readability of code one needs to look at processes around documentation that are established. This is discussed to some degree in our literature review. When comparing students and IT professionals we have determined that students tend to comment in the code rather than using some external method. IT professionals tend to comment in code almost only if the complexity of the code is great. In other words IT professionals rely on more external methods of documentation. This can be derived to a set of predefined procedures established at the company they work at. They also tend to work on larger projects where the need of documentation is greater and where the code is used for a longer time. In order to maintain the code it is beneficial to use external methods of documentation. This will allow for easier maintenance. Another important factor is how the comments are accessed. According to our literature review, the comments needs to be easy to search among and produce quick results based on the user's input.

R5 - What differences is there between students and IT-professionals when it comes to code readability?
The major difference between students and IT professionals are the routines that are in place for documentation. A system that will run for a long time and will need maintenance needs to be well documented in order for good maintainability to occur. The students don't typically build systems that require long term maintainability and therefore they comment directly in the code without any external measures. In the scope of our survey we can clearly see that students will in most, if not all, cases request more documentation than the IT professionals. This shows that the IT professionals has more work experience and knowledge to read someone else's code.

Chapter 7

Conclusion

This thesis shows that code documentation is important within the software industry. Poor documentation can lead to extra costs and time consuming when a piece of code needs refactored, updated or added. In the survey that were conducted, it shows that the IT professionals relays on more architecture and prestudy documentation than the students do. Students on the other hand requires more internal code comments.
There are dozens of ways of writing code comments and documentation but this study shows that group/class comments is still the most popular one.

The correlation analysis with the results from the students and the IT professionals shows that working experience can play a big role in the overall results. Students tend to prefer more comments which can be derived from the fact that they have little to no working experience. It is also possible that students are not familiar with the same standardised method of documentation that the IT professionals are. It is also possible that students today receive better education, where universities focuses on more appropriate code documentation methods. This is however only speculation and have not been derived to the literature in this theses. Students did not comment on the fact that documentation needs to be maintained throughout the lifecycle of the program which the IT professionals in some cases mentioned.

Below are the findings summarized in a list:

- Both students and IT professionals have a good understanding of different programming languages with Java, C++ and C being the most common.
- All participants have some experience. Either professional work experience or university experience.
- Students tend to prefer more documentation than IT professionals.
- The understandability of the code were overall high.
- IT professionals prefer group/class comments.
- Students prefer either group/class comments or self commenting.

Changes

We decided to change the code for question 3 (Appendix A, 1C) to a new one (Appendix A, 1D), because 50% of the students failed this part and 66% thought that the code readability was very bad. We considered the new code to be simpler and had better readability. This was done in order to get a more fair result from the survey and to evaluate the different types of documentation selected in this thesis. The changes were done after the students had done the survey.

Chapter 8

Validity Threats

There are some risks with the survey that is worth mentioning. The participants have different levels of knowledge and different working experiences. There was no action taken to check if a certain participant had extensive knowledge about any specific language.
The different types of languages selected in the survey varies in complexity and could potentially make the results reliable if the users have extensive experience in a certain language. However, it has been established in this thesis that the participants are knowledged in and can use the languages in the survey. In addition to this we look at what comments do to increase the readability and not what the code itself does. A further risk is that we can not control the honesty of the participants in the survey. The amount of participants in the survey are not great and it would be better with more participants in order to get more accurate results. However this is covered by having two controlled groups with an expected level of expertise within the field.
The code examples uses C and Java. We choose these languages because we thought they were most alike and fair in comparison. There is a risk that this decision may have affected the data.
To improve the results the surveys were conducted under controlled circumstances where both time were logged and the surveys were conducted on two separate groups. We could also go much deeper into the literature research. However since the field is so large we had to limit the literature research to something more manageable.

Chapter 9

Future Work

It would be interesting to conduct the survey on a larger scale across different companies. This thesis have determined that processes within the company highly influence the way the software developers write their documentation. It would therefore be of great interest to look more at the complete process rather than a smaller part.

Acknowledgement

We would like to thank our advisor Kennet Henningsson for his support during this work. We would like to acknowledge the time spent reading and discussing this thesis. The feedback we received helped us in our work and is highly valued.
We would also like to thank the students for partaking in the survey and taking time to give us feedback on the survey.
We would like to thank Anton Tell for helping beta-test the survey and for help proofreading. Furthermore we would like to extend our gratitude to the local consultant company that allowed us to sent the survey to their development division.

References

[1] Dale, Edgar and Jeanne S. Chall. 1949. "The concept of readability." Elementary English 26:23. 02-04-2016

[2] Diomidis Spinellis 2010 “Code Documentation” http://ieeexplore.ieee.org.miman.bib.bth.se/stamp/stamp.jsp?tp=&arnumber=5484109 02-04-2016

[3] Daniela Steidl, Benjamin Hummel, Elmar Juergens. 20-21 May 2013. "Quality Analysis of source Code Comments" http://ieeexplore.ieee.org.miman.bib.bth.se/xpl/articleDetails.jsp?arnumber=6613836&tag=1 06-05-2016

[4] Sergio Cozzetti B. de Souza, Nicolas Anquetil, Káthia M. de Oliveira. 21-09-2005. "A study of the documentation essential to software maintenance" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=1085331 06-05-2016

[5] Jef Raskin. 02-03-2005. "Comments are More Important than Code" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=1053354&CFID=781418029&CFTOKEN=81040544 06-05-2016

[6] Jinhan Kim Pohang, Sanghoon Lee Pohang, Seung-Won Hwang Pohang,Sunghun Kim Hong Kong. 01-01-2013. "Enriching Documents with Examples: A Corpus Mining Approach" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=2414783&CFID=781418029&CFTOKEN=81040544 10-05-2016

[7] Peter J. DePasquale, Michael E. Locasto, Lisa Kaczmarczyk and Mike Martinovic 06-10-2012 “// TODO: Help students improve commenting practices” http://ieeexplore.ieee.org.miman.bib.bth.se/xpls/icp.jsp?arnumber=6462504 10-05-2016

[8] Edmund Wong, Taiyue Liu, and Lin Tan 06-03-2015 “CloCom: Mining Existing Source Code for Automatic Comment Generation” http://ieeexplore.ieee.org.miman.bib.bth.se/xpls/icp.jsp?arnumber=7081848 10-05-2016

[9] Philipp Schugerl, Juergen Rilling, Philippe Charland 26-09-2009 “Beyond generated software documentation — A web 2.0 perspective” http://ieeexplore.ieee.org.miman.bib.bth.se/xpls/icp.jsp?arnumber=5306385 10-05-2016

[10] Daniel Schreck, Valentin Dallmeier, Thomas Zimmermann. 2007. "How documentation evolves over time" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=1294952&CFID=781418029&CFTOKEN=81040544 10-05-2016

[11] M. Visconti, C. R. Cook. 2002. "An overview of industrial software documentation practice". http://ieeexplore.ieee.org.miman.bib.bth.se/xpl/articleDetails.jsp?arnumber=1173192&reload=true&action=search&sortType=&rowsPerPage=&searchField=Search_All&matchBoolean=true&queryText=(An%20overview%20of%20industrial%20software%20documentation%20practice) 10-05-2016

[12] https://www.mysql.com/ 10-05-2016

[13] Raymond P.L. Buse, Westley R. Weimer. 2010. "Automatically documenting program changes" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=1859005&CFID=781418029&CFTOKEN=81040544 11-05-2016

[14] David G. Novick, Karen Ward. 2006. "What users say they want in documentation" http://dl.acm.org.miman.bib.bth.se/citation.cfm?id=1166346&CFID=781418029&CFTOKEN=81040544 11-05-2016

[15] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira. "A study of the documentation essential to software maintenance. In ICDC", pages 68–75, 2005.

[16] D. G. Novick and K. Ward. What users say they want in documentation. In Conference on Design of Communication, pages 84–91, 2006.

[17] https://github.com/ 2016-06-20

[18] https://github.com/vim/vim/blob/master/src/netbeans.c 2016-06-20 row 379-435. Copy of the section as changes may acure https://gist.githubusercontent.com/sebbe605/d5feba83bbb0d193816197bd4f63ebab/raw/f66fbbbbf15e63f583b3120081792a7e8678c0fb/foo.c

[19] https://github.com/torvalds/linux/blob/master/tools/iio/iio_event_monitor.c 2016-06-20 row 262-350. Copy of the section as changes may acure https://gist.githubusercontent.com/anonymous/fdcc941d70df18bd72bfb5a5204c8d95/raw/fe4c32e83a8e193c7bb3d06525bae79682bf5c68/gistfile1.txt

[20] https://github.com/greenrobot/EventBus/blob/master/EventBus/src/org/greenrobot/eventbus/SubscriberMethodFinder.java 2016-06-20 row 150-185. Copy of the section as changes may acure https://gist.githubusercontent.com/sebbe605/e537a49b60fa86d98485951d2483c6d8/raw/64e37361f5df4bce2ba73a0f19914dea130c42aa/foo.java

[21] 404 This project have been removed. Copy of the section saved https://gist.githubusercontent.com/anonymous/a4f6f61265a45b319d9c5bbe44319da8/raw/153cc1c1a8f776cdf908b7a2d487ce28727d677e/foo.java

[22] http://www.extremeprogramming.org/ 2016-06-21

Appendix A

Code used in survey

1A Code, Group/Class comments [18]

```
/*
 * While there's still a command in the work queue, parse and execute it.
 */
    void
netbeans_parse_messages(void)
{
    char_u     *buffer;
    char_u     *p;
    int         own_node;

    while (nb_channel != NULL)
    {
        buffer = channel_peek(nb_channel, PART_SOCK);
        if (buffer == NULL)
            break;     /* nothing to read */

        /* Locate the first line in the first buffer. */
        p = vim_strchr(buffer, '\n');
        if (p == NULL)
        {
            /* Command isn't complete.  If there is no following buffer,
             * return (wait for more). If there is another buffer following,
             * prepend the text to that buffer and delete this one.  */
            if (channel_collapse(nb_channel, PART_SOCK) == FAIL)
                return;
        }
        else
        {
            /* There is a complete command at the start of the buffer.
             * Terminate it with a NUL.  When no more text is following unlink
             * the buffer.  Do this before executing, because new buffers can
             * be added while busy handling the command. */
            *p++ = NUL;
            if (*p == NUL)
            {
                own_node = TRUE;
                channel_get(nb_channel, PART_SOCK);
            }
            else
                own_node = FALSE;

            /* now, parse and execute the commands */
            nb_parse_cmd(buffer);
            if (own_node)
                /* buffer finished, dispose of it */
                vim_free(buffer);
            else
                /* more follows, move it to the start */
                STRMOVE(buffer, p);
        }
    }
}
```

1B Code, No comments [19]

```
int main(int argc, char **argv)
{
        struct iio_event_data event;
        const char *device_name;
        char *chrdev_name;
        int ret;
        int dev_num;
        int fd, event_fd;

        if (argc <= 1) {
                fprintf(stderr, "Usage: %s <device_name>\n", argv[0]);
                return -1;
        }

        device_name = argv[1];

        dev_num = find_type_by_name(device_name, "iio:device");
        if (dev_num >= 0) {
                printf("Found IIO device with name %s with device number %d\n",
                        device_name, dev_num);
                ret = asprintf(&chrdev_name, "/dev/iio:device%d", dev_num);
                if (ret < 0)
                        return -ENOMEM;
        } else {
                /*
                 * If we can't find an IIO device by name assume device_name is
                 * an IIO chrdev
                 */
                chrdev_name = strdup(device_name);
                if (!chrdev_name)
                        return -ENOMEM;
        }

        fd = open(chrdev_name, 0);
        if (fd == -1) {
                ret = -errno;
                fprintf(stderr, "Failed to open %s\n", chrdev_name);
                goto error_free_chrdev_name;
        }

        ret = ioctl(fd, IIO_GET_EVENT_FD_IOCTL, &event_fd);
        if (ret == -1 || event_fd == -1) {
                ret = -errno;
                if (ret == -ENODEV)
                        fprintf(stderr,
                                "This device does not support events\n");
                else
                        fprintf(stderr, "Failed to retrieve event fd\n");
                if (close(fd) == -1)
                        perror("Failed to close character device file");

                goto error_free_chrdev_name;
        }

        if (close(fd) == -1)  {
                ret = -errno;
                goto error_free_chrdev_name;
        }

        while (true) {
                ret = read(event_fd, &event, sizeof(event));
                if (ret == -1) {
                        if (errno == EAGAIN) {
                                fprintf(stderr, "nothing available\n");
                                continue;
                        } else {
                                ret = -errno;
                                perror("Failed to read event from device");
                                break;
                        }
                }

                if (ret != sizeof(event)) {
                        fprintf(stderr, "Reading event failed!\n");
                        ret = -EIO;
                        break;
                }

                print_event(&event);
        }

        if (close(event_fd) == -1)
                perror("Failed to close event file");

error_free_chrdev_name:
        free(chrdev_name);

        return ret;
}
```

1C Code, Self-documenting [20]

```
private void findUsingReflectionInSingleClass(FindState findState) {
        Method[] methods;
        try {
            // This is faster than getMethods, especially when subscribers are fat classes
like Activities
```

```
            methods = findState.clazz.getDeclaredMethods();
        } catch (Throwable th) {
            // Workaround for java.lang.NoClassDefFoundError, see
https://github.com/greenrobot/EventBus/issues/149
            methods = findState.clazz.getMethods();
            findState.skipSuperClasses = true;
        }
        for (Method method : methods) {
            int modifiers = method.getModifiers();
            if ((modifiers & Modifier.PUBLIC) != 0 && (modifiers & MODIFIERS_IGNORE) == 0)
{
                Class<?>[] parameterTypes = method.getParameterTypes();
                if (parameterTypes.length == 1) {
                    Subscribe subscribeAnnotation = method.getAnnotation(Subscribe.class);
                    if (subscribeAnnotation != null) {
                        Class<?> eventType = parameterTypes[0];
                        if (findState.checkAdd(method, eventType)) {
                            ThreadMode threadMode = subscribeAnnotation.threadMode();
                            findState.subscriberMethods.add(new SubscriberMethod(method,
eventType, threadMode,
                                    subscribeAnnotation.priority(),
subscribeAnnotation.sticky()));
                        }
                    }
                } else if (strictMethodVerification &&
method.isAnnotationPresent(Subscribe.class)) {
                    String methodName = method.getDeclaringClass().getName() + "." +
method.getName();
                    throw new EventBusException("@Subscribe method " + methodName +
                            "must have exactly 1 parameter but has " +
parameterTypes.length);
                }
            } else if (strictMethodVerification &&
method.isAnnotationPresent(Subscribe.class)) {
                String methodName = method.getDeclaringClass().getName() + "." +
method.getName();
                throw new EventBusException(methodName +
                        " is a illegal @Subscribe method: must be public, non-static, and
non-abstract");
            }
        }
    }
```

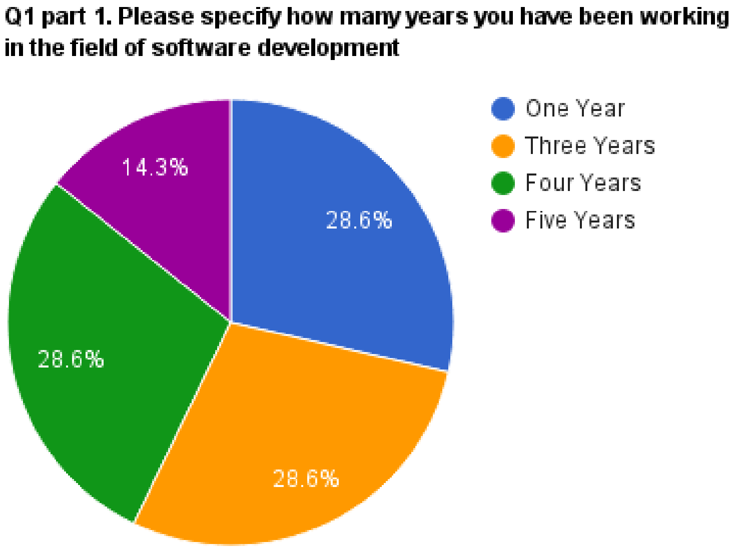
1D Code, Self-documenting [21]

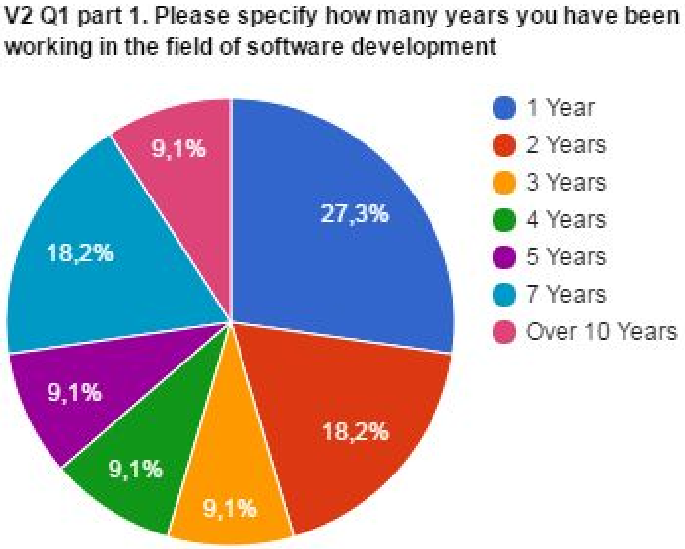
```
public SSLContext build() throws GeneralSecurityException {

    char[] password = "password".toCharArray();
    KeyStore keyStore = newEmptyKeyStore(password);

    if (chain != null) {
      Certificate[] certificates = new Certificate[chain.length];
      for (int i = 0; i < chain.length; i++) {
        certificates[i] = chain[i].certificate;
      }
      keyStore.setKeyEntry("private", chain[0].keyPair.getPrivate(), password,
certificates);
    }

    for (int i = 0; i < trustedCertificates.size(); i++) {
      keyStore.setCertificateEntry("cert_" + i, trustedCertificates.get(i));
    }

    KeyManagerFactory keyManagerFactory =
        KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
    keyManagerFactory.init(keyStore, password);
    TrustManagerFactory trustManagerFactory =
        TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
    trustManagerFactory.init(keyStore);
    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(),
        new SecureRandom());
    return sslContext;
  }
```

Appendix B

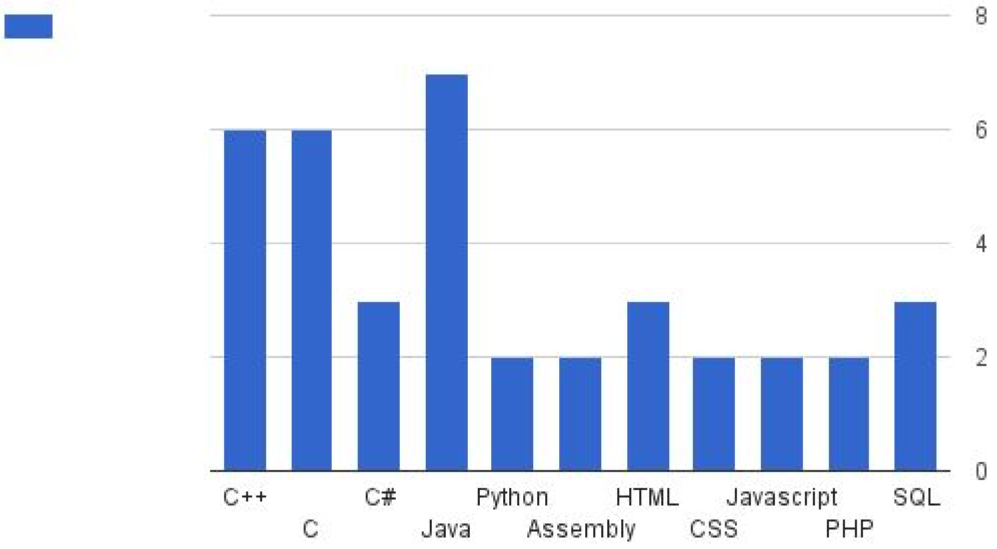
Data from survey

Q1 Pre Questions

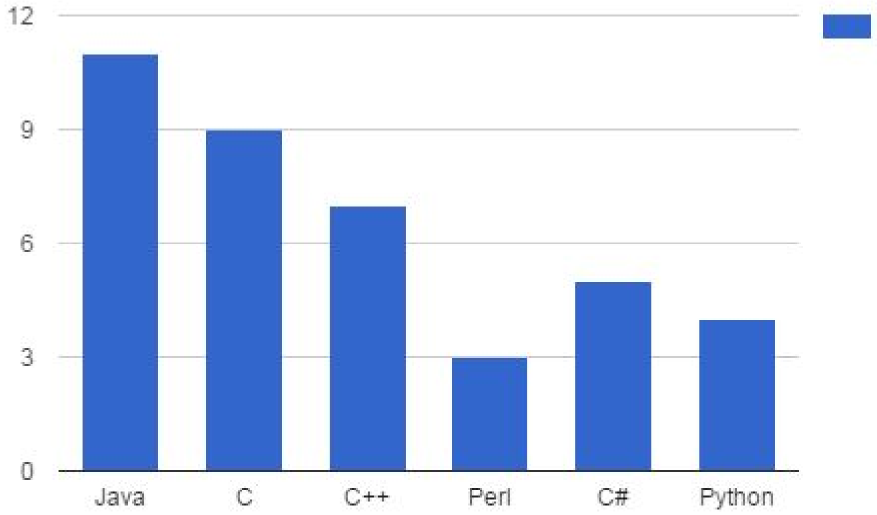




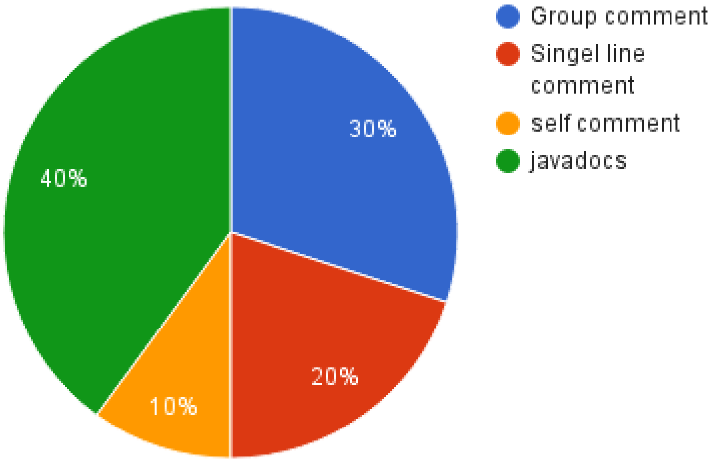
Q1 part 2. Please enter those programming languages that you have knowledge about and can use.



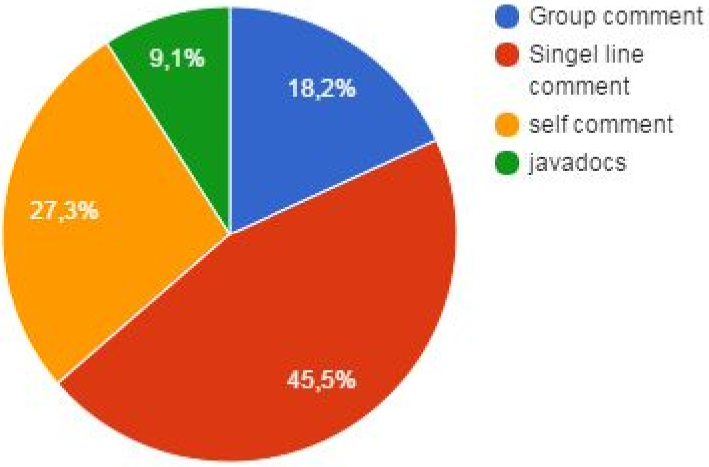
V2 Q1 part 2. Please enter those programming languages that you have knowledge about and can use.



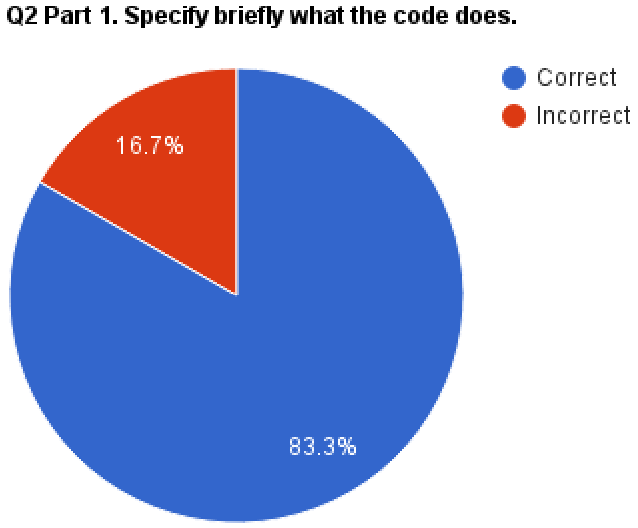
Q1 part 3. How do you comment your code and what do you consider to be a good standard of documentation?

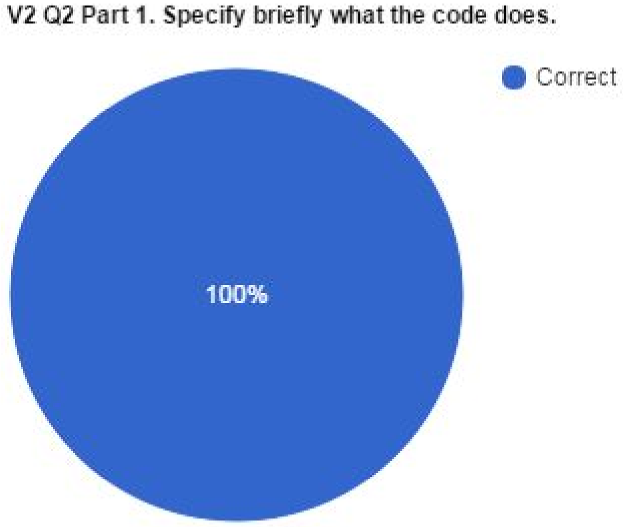


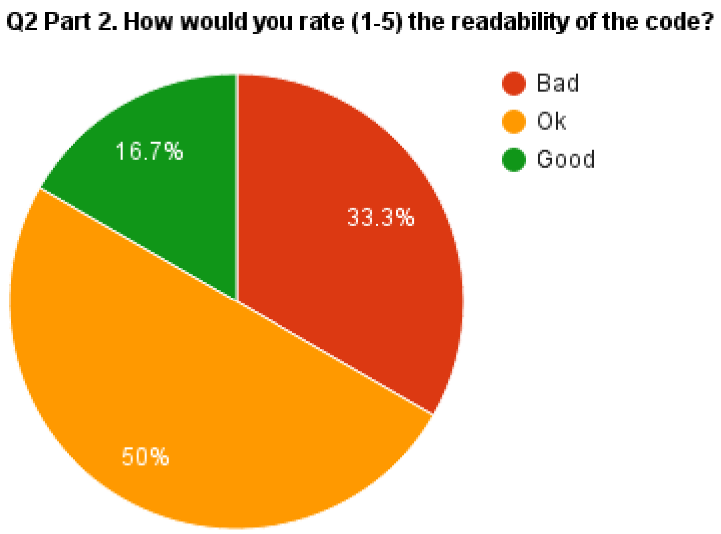
V2 Q1 part 3. How do you comment your code and what do you consider to be a good standard of documentation?



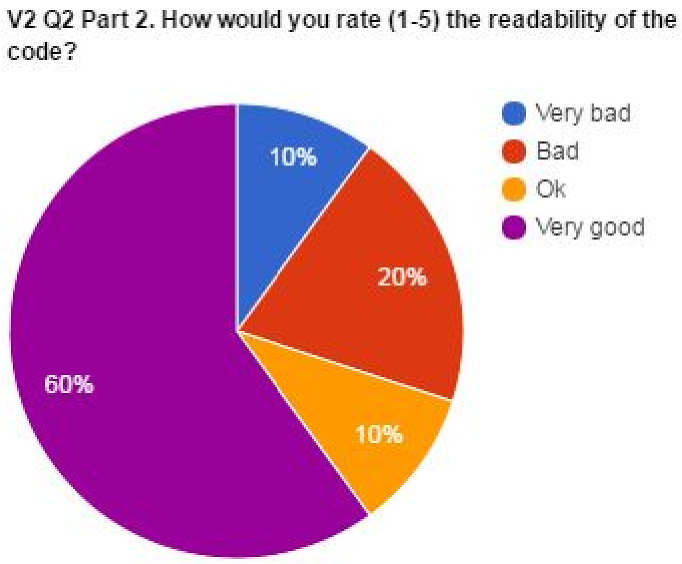
Q2 Group/Class Comments







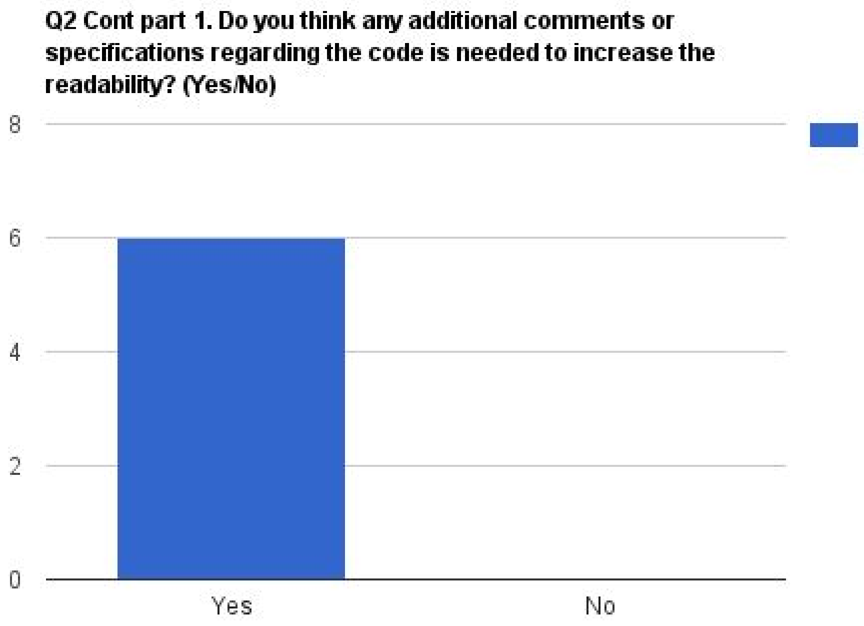
Q2 Part 2. How would you rate (1-5) the readability of the code?

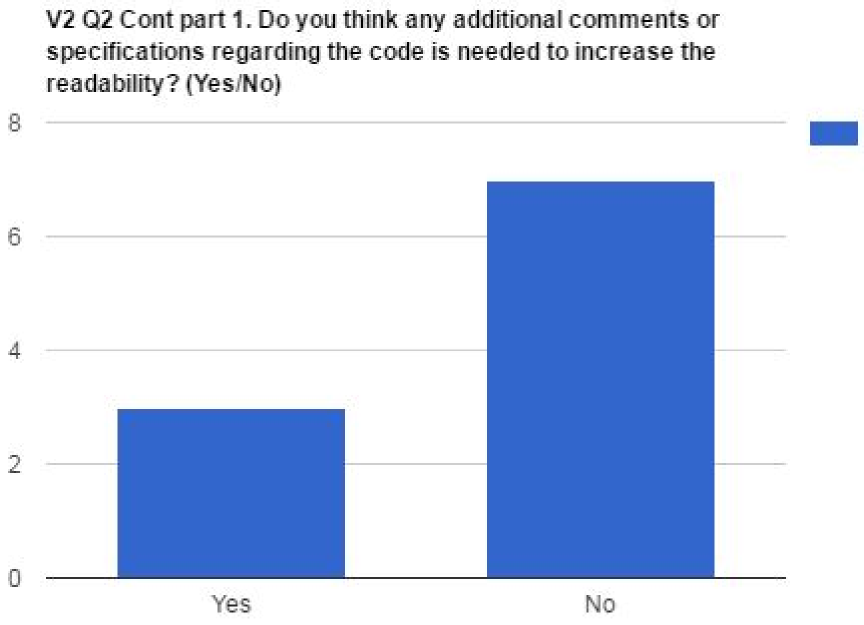


V2 Q2 Part 2. How would you rate (1-5) the readability of the code?

Q2 Part 3. Add comment (not required).

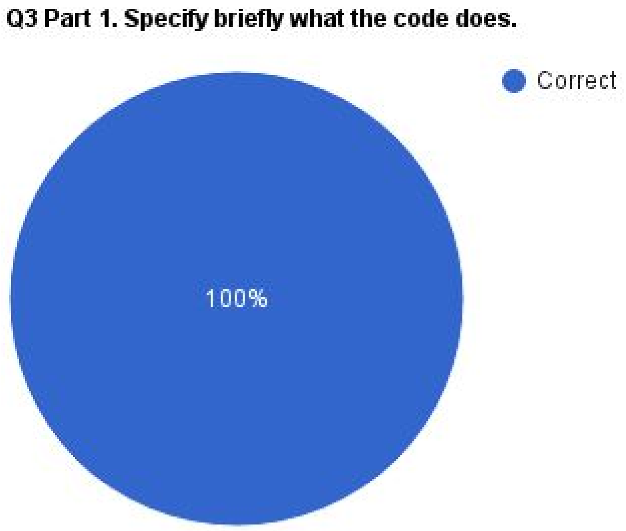
Q2 Cont

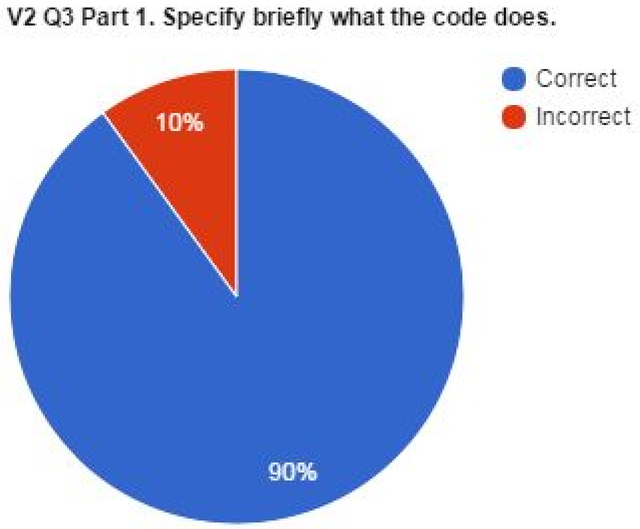




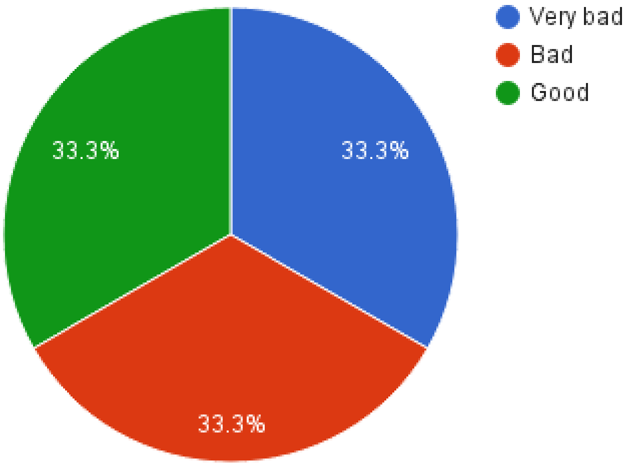
Q2 Cont part 2. Add comment (not required).

Q3 No Comments

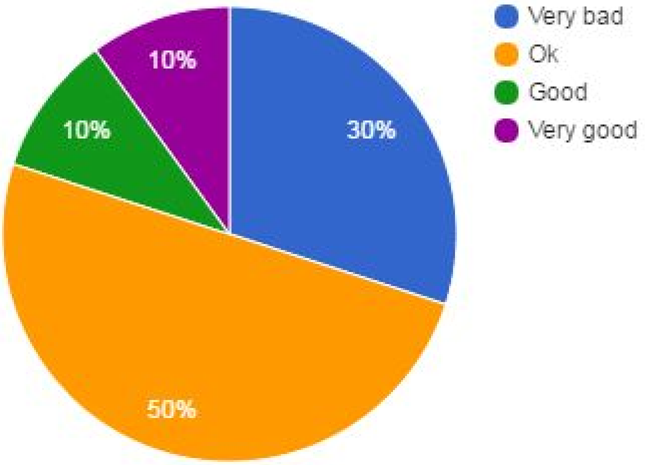




Q3 Part 2. How would you rate (1-5) the readability of the code?

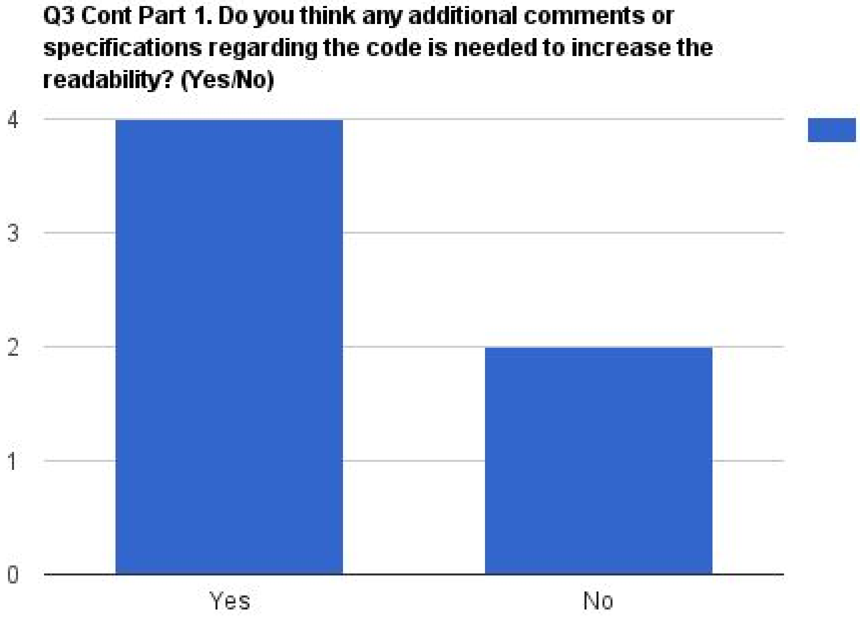


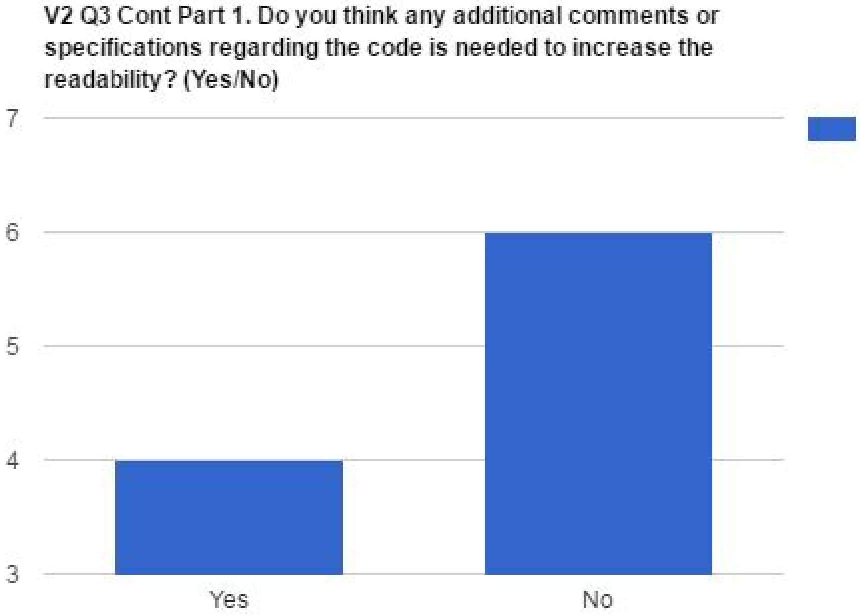
V2 Q3 Part 2. How would you rate (1-5) the readability of the code?



Q3 Part 3. Add comment (not required).

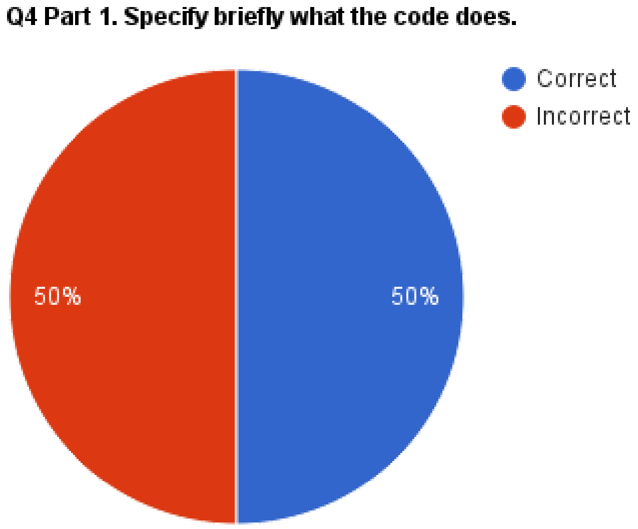
Q3 Cont

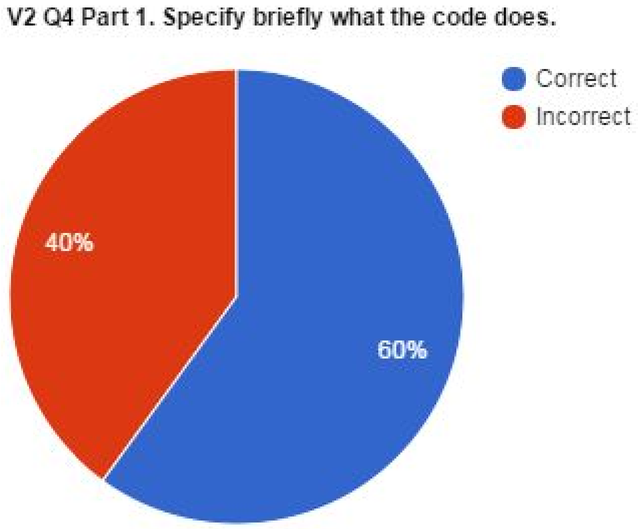


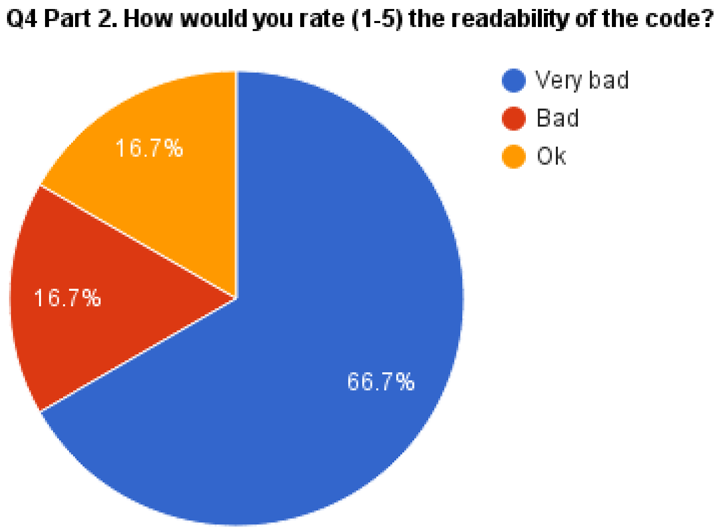


Q3 Cont Part 2. Add comment (not required).

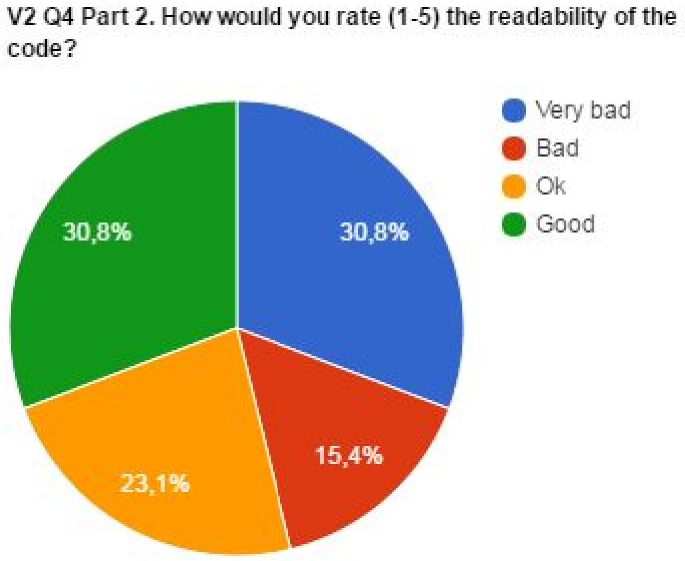
Q4 Self Documenting







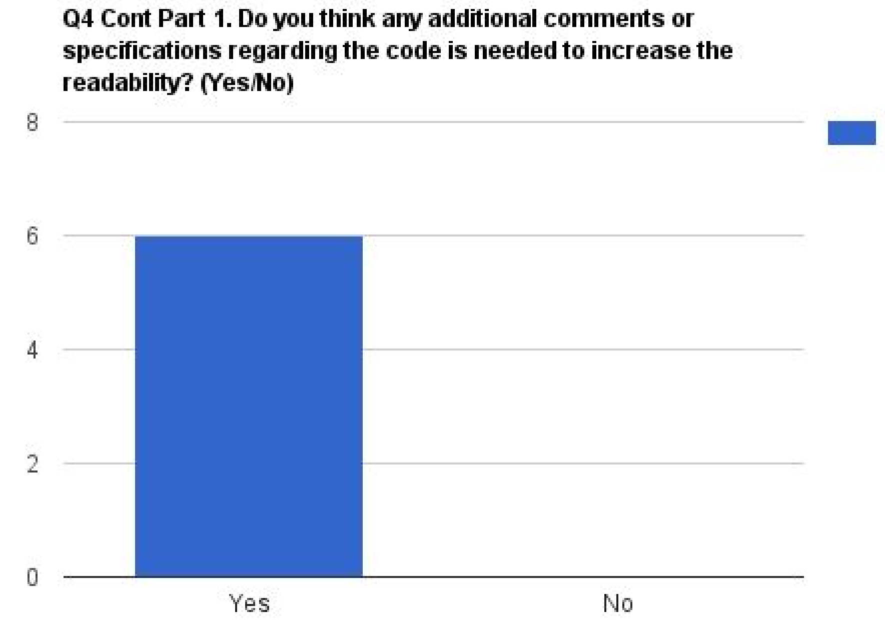
Q4 Part 2. How would you rate (1-5) the readability of the code?

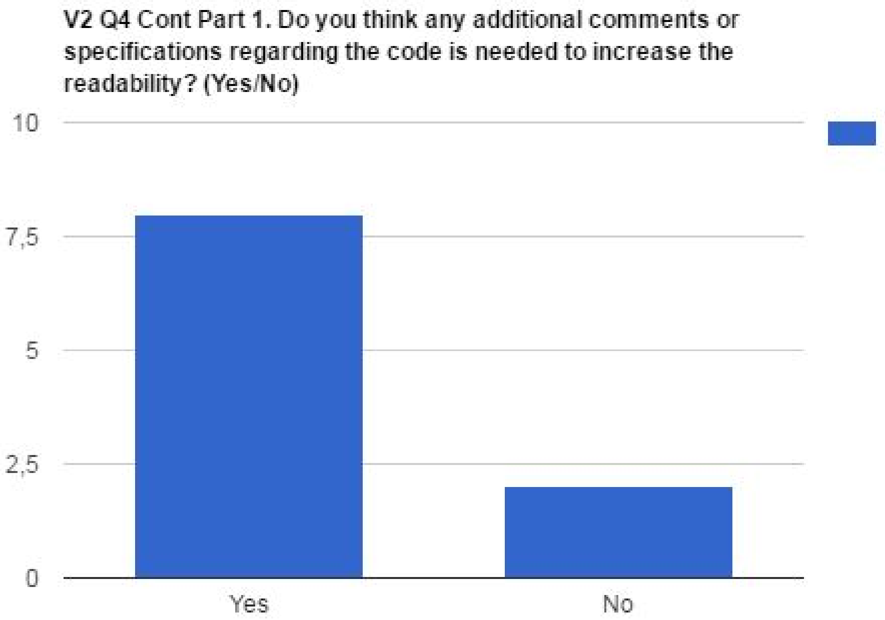


V2 Q4 Part 2. How would you rate (1-5) the readability of the code?

Q4 Part 3. Add comment (not required).

Q4 Cont

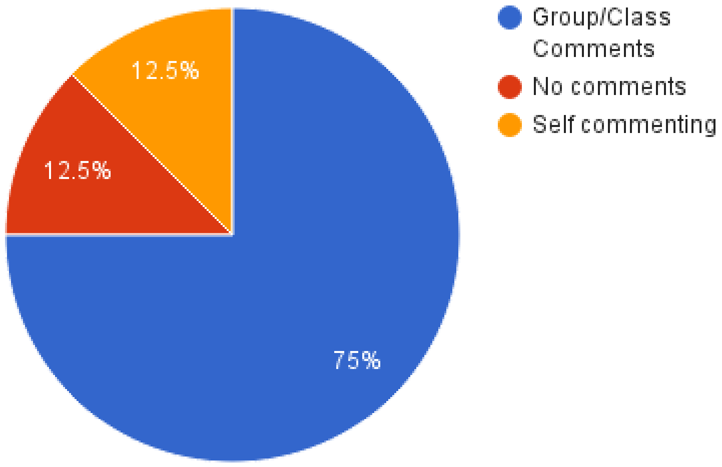

Q4 Cont Part 1. Do you think any additional comments or specifications regarding the code is needed to increase the readability? (Yes/No)


V2 Q4 Cont Part 1. Do you think any additional comments or specifications regarding the code is needed to increase the readability? (Yes/No)

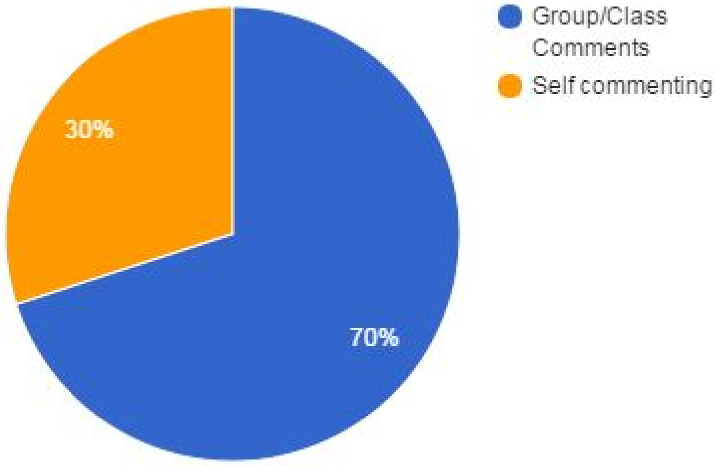
Q4 Cont Part 2. Add comment (not required).

Q5 Final questions

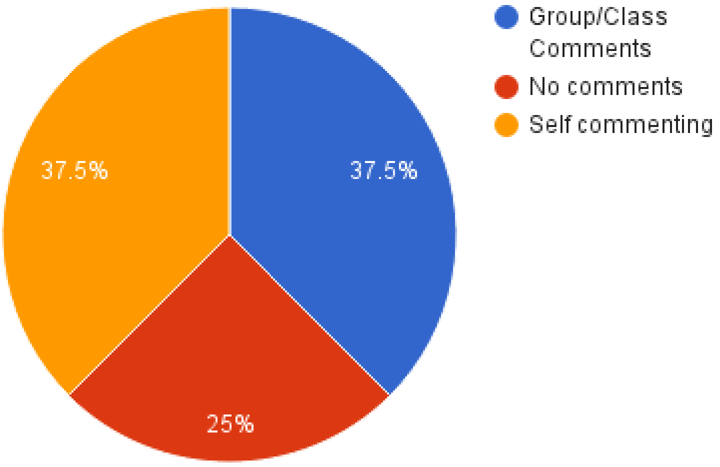
Q5 Part 1. Which of the three alternatives shown in this survey do you prefer?



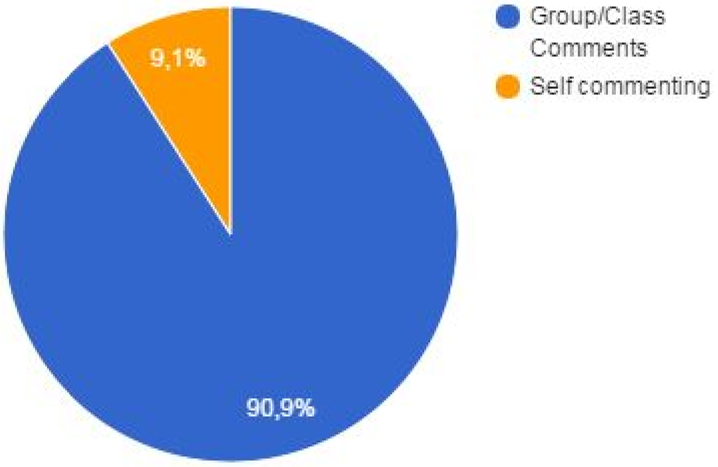
V2 Q5 Part 1. Which of the three alternatives shown in this survey do you prefer?



Q5 Part 2. Which of the three alternatives shown in this survey do you think takes the longest time to write?



V2 Q5 Part 2. Which of the three alternatives shown in this survey do you think takes the longest time to write?



Appendix C

Comments from survey and links to source data

Data from textboxes (How do you comment your code and what do you consider to be a good standard of documentation?)

IT professionals

- by section unless some specific line is of extra interest and importance
- "Don't really understand what you are getting at here…
Javadoc for classes and public methods"
- "As much as humanly possible, I don't.
The only 2 cases where I think comments are ok: a header to a file that explains what the file/class is for, or on configurations that need to explain the impact of what it means.
Most other cases you have either a class that does way to much, a method that is too large, or just don't care to name your methods and variables."
- line by line
- description at start of method/function and lines when needed
- rarely at all
- small notes in the sections where it's needed
- As little as possible. We only document public methods on an interface level. In-code comments is only used if there is a particular need for it (very complex algorithms, code that is hard to understand), but that is very rare. If you write Clean Code that is easy to follow, the code shouldn't need extra comments that will need maintenance.
- I write comments by section, grouping it up with relevant code sections
- a comment at the start of each method as well as comment lines when needed to explain particular passages in the code
- one introduction at the top of a function and a line of needed around specific code segments
- Links to source data

Students

- I comment above each function, and above each code line that may be hard to understand. I try to use javadoc (whenever possible) commenting style in my line of work.
- I prefer to use good naming conventions for variables and methods than have long comments. I might preface a block of code or a method with a short sentence describing its function.

- Short comment about function input/output (auto generated), comments to the right of the code
- "/**
  * comment
  *
  *
  */
- // comment"
- "I'm not very good at documenting or commenting.
  Javadoc style of in-file code comments. "
- I usually describe what the function does, along with explanations of the parameters and return values. in the style of javadocs usually. When someone describes it so that I know how and what a function requires is when I find the code standard to be good.
- "I comment only on parts where i feel that the code can be interpreted in multiple ways. I do this to avoid misunderstandings.
  I think the standard to aim towards should be readable code and as little comments as possible"

Link to source data

The Data in its raw format can be found on Google Drive in PDF format.
Link one students:
https://drive.google.com/file/d/0B-spKeE7CTf4emdWaVN1MVQtYmc/view?usp=sharing
Link two IT professionals:
https://drive.google.com/file/d/0B-spKeE7CTf4U09yekh0c0dHZUU/view?usp=sharing