EREN YILDIZ

M.S. Thesis

2017

AUTOMATIC COMMENT GENERATION USING THE SOURCE CODE

EREN YILDIZ

IŞIK UNIVERSITY

2017

AUTOMATIC COMMENT GENERATION USING THE SOURCE CODE

EREN YILDIZ
B.S., Software Engineering, IŞIK UNIVERSITY, 2015
Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2017

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

AUTOMATIC COMMENT GENERATION USING THE SOURCE CODE

EREN YILDIZ

APPROVED BY:

Assist. Prof. Emine Ekin (Thesis Supervisor)	Işık University
Assoc. Prof. Olcay Taner Yıldız	Işık University
Assist. Prof. Reyhan Aydoğan	Özyeğin University

APPROVAL DATE: 23/01/2018

AUTOMATIC COMMENT GENERATION USING THE SOURCE CODE

Abstract

In this study, automatic comment generation for Java methods is described. It is sufficient that the codes conform to the syntax rules of the Java programming language, and it is not expected to be runnable. In order to generate comments, source code is examined syntactically. At this stage, only the method signature and its return type is needed. By working on open source Java projects, different templates have been developed for different method types. Using the compiled information which is the result of the examining source code that is currently being developed, the most suitable template is chosen and texts are created. These texts explain the aim of the method. Created texts are added to source code as a comment.

Keywords: source code summarization, documentation generation, program comprehension

KAYNAK KOD KULLANARAK OTOMATİK YORUM OLUŞTURMA

Özet

Bu çalışmada, kaynak kodlara metot seviyesinde yorum ekleme işinin otomatikleştirilmesi anlatılmaktadır. Kodların, Java programlama dilinin sözdizim kurallarına uygun olması yeterli olup, çalışabilir durumda olması beklenmemektedir. Yorum üretmek için kaynak kod biçimsel açıdan incelenir. Bu aşamada ilgili metodun sadece imzasına ve geri döndürdüğü veri tipine ihtiyaç duyulur. Açık kaynak kodlu Java projeleri üzerinde yapılan çalışmayla farklı metot türleri için farklı şablonlar geliştirilmiştir. Yazılımcının geliştirdiği kodun incelenmesi sonucu derlenen bilgi ile bu şablonlardan en uygun olanı seçilir ve metinler oluşturulur. Bu metinler metodun amacını açıklar. Oluşturulan metinler yorum olarak kaynak koda eklenmektedir.

Anahtar kelimeler: kaynak kodun özetlenmesi, dökümantasyon üretme, program anlama

Acknowledgements

To My Family...

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	**A**pplication **P**rogram **I**nterface
AST	**A**bstract **S**yntax **T**ree
IDE	**I**ntegrated **D**evelopment **E**nvironment
POS	**P**art **O**f **S**peech
BNF	**B**ackus-**N**aur **F**orm
LHS	**L**eft **H**and **S**ide
RHS	**R**ight **H**and **S**ide

Chapter 1

Introduction

Comments are needed for developers to adapt to the code that they have never seen before or worked on it for a long time ago. It is essential to capture the most important parts of the source code and then translate them to proper English sentences to give developer hints about what the related method does without even skimming it. This thesis proposes a new framework named Autocomment to create comments using the source code.

Autocomment is a framework to create comments using the method signature and the source code in their body. In other words, Autocomment is a translator which translates source code to a natural language sentence.

All previous works on automatic comment generation, aim to replace hand written javadocs with automatically generated ones unlike Autocomment. As will be explained in section 1.3, Autocomment does not only aim to create comments which can only be read by developers but all other people who are just started to programming or working in programming industry such as managers, project leaders, researchers, etc. Autocomment achieves this by adding new techniques to literature such as; simplified data types, more dense comments which hosts multiple code lines that do the same work, simpler comments to inform non-programmers about the project, a recursive structured algorithm to achieve the ability to generate comment for every element in syntax and etc. Survey results in the Chapter 4 justifies that Autocomment is able to achieve these goals.

Autocomment utilizes Abstract Syntax Tree(AST) to extract information from source code to use in comment generation process. In the section 1.1, how Autocomment utilizes AST is given alongside with the definitions of syntax and grammar.

Autocomment assumes developers to name their variables, methods and classes meaningfully and intuitively to be able to generate informative comments and tokenize the given strings. In the section 1.2, code conventions in Java, mainly the naming convention is explained.

After that, aims and benefits of Autocomment is explained in detail and finally how this thesis is organized is given.

1.1 Syntax, Grammar and AST Definition and Utilization

Syntax is the set of all rules to specify which combinations of symbols in given order is accurate for the language. Grammar on the other hand, is a representation of how syntax for a particular language should be formed. In Code block 1.1, an example rule is given for Java grammar[1, 2] which specifies how conditional statements should be written by a developer.

```
1   IfThenStatement: if ( Expression ) Statement
```

Code Block 1.1: IfThenStatement rule in Java grammar

Rules get their name from the variable from left hand side(LHS). Since LHS can only have one variable, rule's name always will come from the variable on the LHS. On the right hand side(RHS), there can be variables or terminals. In Code block 1.1, there is a rule named "IfThenStatement". Terminals are "if", "(" and ")". Variables are "IfThenStatement" on LHS, "Expression" and "Statement" on RHS which have their own rules for productions.

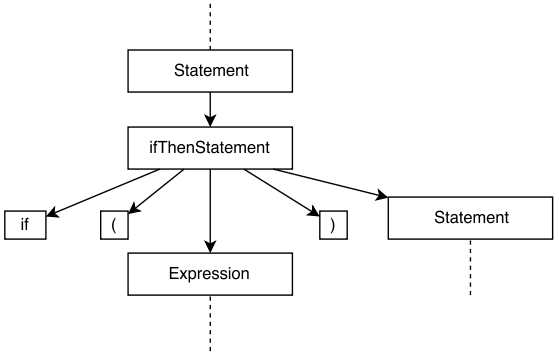


Figure 1.1: Abstract Syntax Tree Representation of If statement

In order to use information that lies in grammar, an implementation of it is needed. Abstract Syntax Tree(AST) is the implementation for grammars of programming languages to represent syntactic structure of the source code in a tree. In Figure 1.1, an example AST representation of Code block 1.1 is given to show how AST may look like. In AST, each node in tree represents a symbol in grammar. For example, an if statement in the body of a method would imply that if statement is actually a child node of the method's body statement. AST doesn't have to show everything like in the language's syntax as the term "abstract" that is used in AST implies it. Autocomment utilizes AST to get data from source code such as; such as method's name, method's body, parameters, statements and etc. to be used in comment generation process.

1.2 Code Conventions for Java

Code conventions are the guidelines which help developers to ensure for code to be efficient, consistent and clear. For example, in Java, variable and method names should follow CamelCase notation as convention. Other convention rules are including but not limited to[3];

- Class and Interface names starts with a capital letter
- Letters in package names are written in lower case
- Method names contain at least one verb

Code conventions are important because [3]:

- It makes maintenance easier
- It makes it convenient for a team of developers to work on same project
- It increases readability
- Make sure the product followed a standard and it's well-packaged and clean

One of the code convention types is the naming convention. Naming convention consists the guidelines to help developers to name their identifiers consistently such as; classes, interfaces, methods, variables and constants. By the naming conventions in Java, developers are expected to use CamelCase notation as much as possible except for constants(which should be all capital letters) and commonly used abbreviations(like URL, HTML, etc.).

1.3 Aims and Benefits

This thesis's main aim is to generate comprehensible and informative comments as natural language sentences. There are also more specific aims which are being used in the way of completion of the main aim.

One of the aims is to be able to use method signature to generate summary comments by extracting and tokenizing the necessary components out of the method signature. Summary comment is the first comment type that Autocomment can generate. It has various benefits and will be explained further in this section.

Another aim is to be able to find the most influential code lines in the method body and generate important statements comment out of it. Important statements comment is the second comment type that Autocomment can generate. As summary comments, important statements comment's benefits will also be explained further in this section.

Achieving the aims, users may benefit from Autocomment in several ways.

One of the benefits is that, it provides a quick glimpse of what the project is capable of without tracing, running or debugging the code itself. By doing so, even without the technical skills to understand the source code, these new developers would get the basic idea of project's capabilities and objectives. Autocomment's "summary" comments are for this basic need.

Another benefit is that, it provides natural language sentences to students, newcomers and people who just started to learn programming and at beginner level to let them learn what the code does by themselves. By doing so, instructor doesn't always need to explain the code every bit of the code, in fact, this will let the students experiment the code by themselves. Therefore, Autocomment doesn't only help students by providing simple comments, but also give instructor an advantage of preparing lecture notes, homeworks and projects quickly as the source code will be more explanatory by itself with auto generated comments which follows a pattern.

Programmers who are more experienced can also get benefit of Autocomment by reading "important statements" comments. This type of comments are indeed has more depth than the summary comments and having technical skills is necessary to understand it. Generally, experienced developers are tend to trace the code over comment reading but when it comes to methods with long body which means lots of lines of code, even the experienced developers have to spend too much time to detect the important statements. Thus, providing an in-depth explanations of the source code to experienced developers is also one of the main purposes of Autocomment.

Another benefit is that, it provides lazy developers who doesn't spend the time needed to write comments. Sometimes developers are tend to be lazy or forgetful which leads to projects with lack of comments and documentation. Especially in the projects which is developed by a group of programmers would need their

source code commented to follow the objectives and finish the project without exceeding the dead line. Autocomment solve this problem by generating comments automatically which makes developers to have more time on writing the source code rather than writing comments.

Last but not least, Autocomment's auto generated comments can be used to create an API documentation. Although Autocomment's major audience is developers, it is not limited to that. Managers and project leaders who have the role of leading and managing rather than writing the source code would need a detailed structure of the source code which explains what project does in its current state to further help to developers. For example, a collection of summary comments can be given to managers and a collection of important statements with summary comments can be given to project leaders. This makes them follow developers actions in an much more quick and efficient way as Autocomment's comment can be updated as the code changes automatically.

Besides the main benefits, Autocomment has lots of small benefits such as leading developers to write a better and cleaner code which includes better variable names and method names, providing leaders a documentation which always have the same standard which also makes it more understandable and intuitive in time.

1.4 Thesis Organization

This thesis is organized as follows. In Chapter 2 the necessity of comments will be explained by surveys and then, the state-of-art techniques which are being used in comment generation will be introduced. The approach will be detailed in Chapter 3. Survey results will be presented in Section 4 and work will be concluded within Section 5.

Chapter 2

Literature Survey

Understanding the source code and generating natural language summaries for the source code is a problem that has been studied over a long time.

Although it is not a work about comment generation, in 2006, Zhang et al. has proposed an approach to identify use cases in source code [4]. This is one of the early steps of how one can understand what the source code does as the very first step towards generating comments using the source code is to understand what the source code does. Their main is that branch statements are the most important structures to distinguish one use case from another in source code. They have designed a static representation of software systems which means it uses design elements instead of run time elements. They also have used call graphs which is an representation of which function calls another function. The software system they have designed incorporates branch information into the traditional call graph, which is named Branch-Reserving Call Graph. As a result, their approach produces combinations of use cases rather than individual use cases occasionally and still needs a human involvement.

Comments are valuable as they contains design decisions and intentions which is delivered to developer and maintainers. [5]. Comments are critical in order to give developers to get to know the source code as fast and accurate as they can. In 2009, Fluri et al. have analysed the co-evolution of comments and source code[5]. They have studied the question whether developers write comments and

at what level developers add comments throughout the lifetime of the developing process. They have presented an approach to associate comments with entities which is created using source code to track comments' co-evolution over multiple versions to answer that question. They have found that amount of source code and comments grows nearly at the same rate; source code entity's type, such as a method declaration or an if statement, has big influence on whether or not it has comments; in six out of the eight systems, comments and source code co-evolve in 90% of the cases; and API changes and comments do not co-evolve but they are re-documented in a later revision. As a result, their approach enables a quantitative assessment of the commenting process in a software system.

In 2010, Sridhara et al. have presented a novel technique to automatically generate descriptive summary comments for Java methods. Their automatic comment generator uses method signature and its body to identify the content and generate summary text that summarizes the method's overall actions [6]. In 2011, Sridhara et al. have extended their work by presenting an automatic technique for identifying code fragments which finds the most informative code lines and express them as a natural language sentences. [7]. Sridhara et al. also have presented heuristics to generate comments that provide a high-level overview of the role of a parameter in a method [8]. Their aim was to generate parameter comments and integrate them with method summaries.

In 2010, Haiduc et al. have presented a study that investigates the suitability of various techniques for generating source code summaries based on position of words or sentences and techniques based on text retrieval[9]. They have implemented text retrieval techniques; one based on Vector Space Model and the other one on Latent Semantic Indexing to create descriptions out of source code.

In 2013, Moreno et al. have created an Eclipse plug-in for automatically generating natural language summaries of Java classes named JSummarizer [10]. The tool uses a set of predefined heuristics to choose which information will be used

in the summary, and it uses NLP and generation techniques to build up the summary. In 2014, Moreno also has proposed an approach to generate summaries of complex code artifacts, such as, classes and change sets [11].

In 2014, McBurney et al. have proposed a technique that includes context by analyzing how the Java methods are invoked [12]. They have used PageRank to choose contextual information to summaries which they compute for the program's call graph. They then build a natural language generation system to interpret the keywords and infer meaning from the context.

This thesis extends our previous work [13] by improving summary comments and adding important statement comments. In our previous work, Autocomment was only able to generate summary comments which is also beneficial to developers but summary comments' main purpose is to help non-developers to track the project besides providing a quick glimpse to developers to inform them what the method does. By adding important statements, developers are now able to learn "how" method works instead of just "what" method does by picking the most critical code lines in method body and representing them as natural language sentences.

Chapter 3

Approach

Autocomment is a framework to generate comments automatically. An overview of the comment generation process can be seen in given Figure 3.1. In order to accomplish this task, it first gets the source code and then preprocess it by extracting, tokenizing, simplifying and postagging the necessary components. After that, Autocomment analyzes method signature and chooses the proper template to generate summary type comments. All of these analysis of methods will be explained in detail in the section 3.1. On the other hand, Autocomment also analyzes the method body and extracts the most important code lines which also be called as EUnits to generate most important statements comments for the given method. All of these analysis of EUnits will be explained in detail in section 3.2.

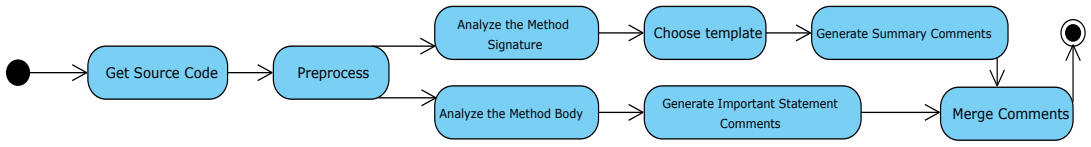


Figure 3.1: Overview of comment generation process

Autocomment uses a rule based approach by utilizing the AST of source code to generate comments for methods. In order to make the most out of the AST, an open-source library called SPOON is used. Spoon enables Java developers to write a large range of domain-specific analyses and transformations in an easy and concise manner [14]. Figure 3.2 shows how SPOON represents AST.

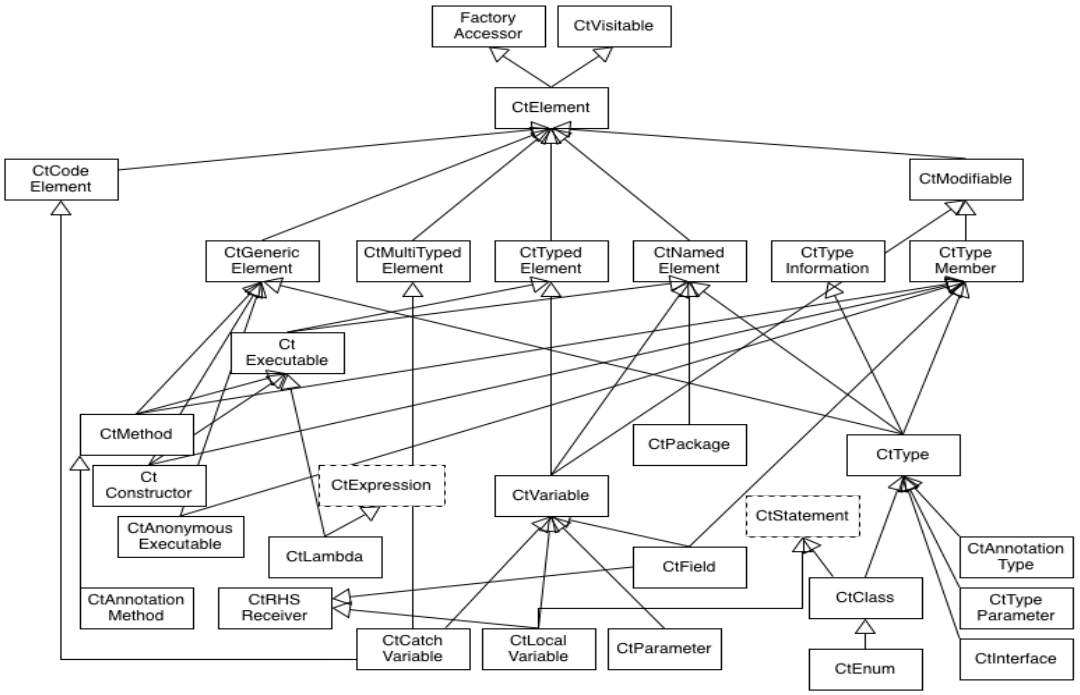


Figure 3.2: Abstract Syntax Tree Representation in SPOON of Java

For example, in the Code block 3.1, a Java class named "Animal" containing a single method is given. Autocomment uses SPOON here to extract AST out of the code in order to use in further comment generation processes. In Figure 3.3, a simplified version of Animal class's AST is given.

```
1  public class Animal{
2     public void eat(Animal e){
3        if (e instanceof Edible){
4           this.weight += e.getWeight();
5           e.sendToGraveyard();
6           System.out.println(this.toString() + " ate " + e.toString());
7        } else {
8           System.out.println(e.toString() + " is not edible");
9        }
10    }
11 }
```

Code Block 3.1: A class example in Java

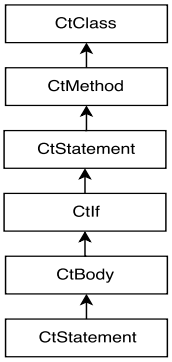


Figure 3.3: Simplified AST of Animal Class

3.1 Analysis of Methods

In order methods to be used in comment generation process, they first needed to be preprocessed. In this section, how Autocomment prepares the methods is explained. This is mainly for summary comments and in the Figure 3.4, the process for summary comment generation process is shown.

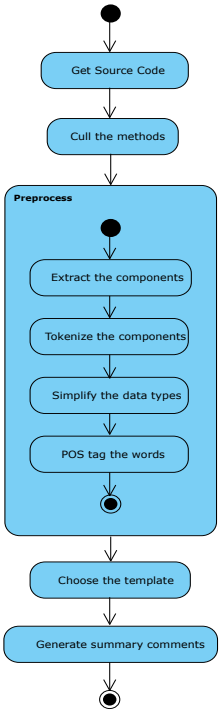


Figure 3.4: Overview of summary comment generation process

3.1.1 Culling the methods

Comments are mandatory for developers to both understand, maintain and implement future releases of project.However, generating comments for every methods is not necessary as some of the methods are comprehensible without comments. In the list below, method variations which are excluded for comment generation process are given:

- Accessors & Mutators(a.k.a getters & setters)

  Accessors and mutators are the methods that controls access to related class's private fields. Nowadays, most of the IDE's such as Eclipse, IntelliJIDEA, NetBeans etc. have an option to generate accessors and mutators automatically. Having the same formula syntactically for accessors and mutators relatively makes IDE's possible to generate them accurately. In fact, they always have only one line of code in body and have same formula, makes them understandable intuitively. Thus, accessors and mutators are excluded from comment generation process.

- Methods that inherited from Object class(e.g. toString(), equals())

- Interface methods(e.g. compareTo())

  Interface methods and the methods like toString(), equals() etc. that inherited from Object class, although may have different code in their body, the purpose of the implementation of these methods is always same. Thus, these variety of methods are not part of the comment generation process and will not be commented by Autocomment.

- Unit test methods

  Unit test methods are the methods that tests a method, process or system to check whether the system is robust and works properly or not. Having the same purpose of writing unit methods, makes them insignificant to be commented. Therefore, they are excluded.

- Methods with empty body

  Methods with empty body are the methods are not yet completed. It is better for Autocomment to skip these methods as skipping them makes Autocomment to finish the job faster. Therefore, they are excluded.

3.1.2 Extraction of components in method signature

Two of the components of a method declaration comprise the method signature the method's name and the parameter types[15]. An example of method signature is given in Code block 3.2.

```
1   equalsIgoreCase(java.lang.CharSequence, java.lang.CharSequence){...}
```

Code Block 3.2: Example of a method signature

In this state, method's components which are method name and parameters are extracted from method's signature. For the method signature that is given above, method name is "equalsIgnoreCase" and the parameters are "java.lang.CharSequence, java.lang.CharSequence". These components will be further processed as explained in the following subsections.

3.1.3 Tokenizing the components by CamelCase and snake_case notations

In programming, CamelCase and snake_case notations are the most frequently used naming conventions. CamelCase notation is an naming convention that works in a way that the first word consists of lower case letters and the other words has an uppercase letter as first letter and lower case letters until the end of each word(e.g. equalsIgnoreCase). snake_case notation is an another naming convention that follows the rule of having all lower case letters with the words of splitted by underscore. In order to components be used in commments, they must be tokenized by the notation of CamelCase and snake_case. As an example,

for the method signature that is given in Code block 3.2 both method name and parameter components needs to be tokenized. The method name "equalsIgnoreCase" will be tokenized as "equals", "ignore", "case" words and the parameters will be tokenized as "char", "sequence" and the parameter name that doesn't exist in method's signature by definition. Parameter's name also will be used as it may make the comment more informative. For the parameter "CharSequence old_char", the parameter name "old_char" will be tokenized as "old" and "char" words. All the words that is acquired by the process of tokenization will be used both as a word in comments and in the next subsections as they possess many more information like POS tags which also has an important role for comment template selection.

3.1.4 Simplifying the data types

Main aim of the Autocomment is to provide comprehensible comments to developers. However, it is also critical to make comments informative. So as to keep the balance of comments between informative and comprehensible, a simplification process is required. There are 2 types of simplifications in Autocomment; simplifying the collections and maps data types and removing the package names at the start of data type.

A collection -sometimes called a container- is simply an object that groups multiple elements into a single unit[16]. Collections are used to store, retrieve, manipulate and communicate aggregate data[16]. Collections are frequently used in programming but to give developers easy to ready comments, a simplification is required. An array list initialization code with the name of "usernames" is given in Code block 3.3.

In order to simplify it, its punctuations are removed. Moreover, to make it looks even simpler, all data types that inherited from collection framework will be simplified as "Collection of usernames".

```
1   ArrayList<String> usernames = new ArrayList<>();
```

Code Block 3.3: ArrayList initialization example in Java

In addition, if it has more than one dimension, as the ArrayList with the name of "userAddresses" given in Code block 3.4, its comment will also have the information about its dimension as its comment would be "2D Collection of user addresses". Simplification process for the map types and the types that is inherited from map class follows the same convention as collections.

```
1   ArrayList<ArrayList<String>> userAddresses = new ArrayList<>();
```

Code Block 3.4: Two dimensional ArrayList initialization in Java

Removing the package names is the other simplification for data types. For example, "java.util.String" will be transformed to "string".

3.1.5 POS tagging the words

Autocomment uses Stanford's CoreNLP which uses Penn Treebank POS tags that is given in Table 3.1 to postag the words. The process of setting the postags for word is called pos tagging. Stanford CoreNLP is a framework that covers most of the common natural language operations.[17]. In Figure 3.5, an example of postagging a method's name process is given.

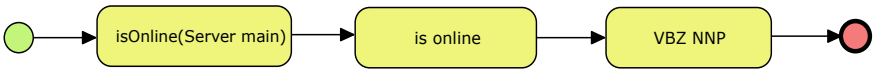


Figure 3.5: Example of postagging a method's name

Postags will be used to choose which comment template will be used to generate comments.

Number	Tag	Description
1	CC	Coordinating Conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or present participle
30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Possessive wh-pronoun
36	WRB	Wh-adverb

Table 3.1: Penn Treebank POS Tags[18]

3.2 Analysis of EUnits

EUnits are the most important statements to be used in comment generation process. In Figure 3.6, overview of important statements comment generation process is shown. Although type names of EUnits are same (except for Same Action Sequence EUnit, they are stated as Same Action SUnit in [6]), their capabilities and how Autocomment selects them varies. Those differences will be stated in the following subsections.

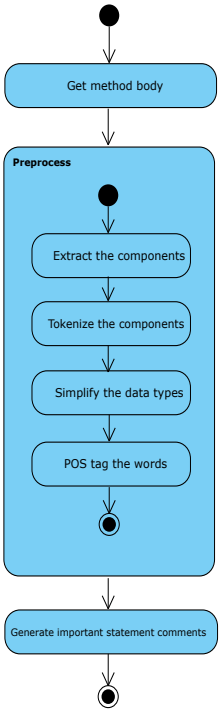


Figure 3.6: Overview of important statements comment generation process

3.2.1 Identifying the EUnits

There are 5 type of EUnits; Ending EUnits, Void Return EUnits, Same Action Sequence EUnits, Data Facilitator EUnits and Controlling EUnits. Autocomment uses SPOON to take advantage of AST to acquire statements in order to create EUnits.

```
1   public Component buildMainWindow(List<Component> components){
2      Component mainComponent = new Component("Main Window");
3      mainComponent.addComponent(new Button("Yes"));
4      mainComponent.addComponent(new Checkbox("I accept terms"));
5      mainComponent.addAllComponents(components);
6      if (mainComponent.hasParent()){
7         Component parent = mainComponent.getParent();
8         return parent;
9      } else
10        return mainComponent; }
```

Code Block 3.5: A method example in Java

Ending EUnits are basically the statements that breaks the flow of code such as; return statements, exception throws or if the method has a void return type then the last statement would be ending statement. In the Code block 3.5, 8. and 10. code lines are Ending EUnits.

Void Return EUnits are the statements that doesn't return anything or assign anything. They are mainly inherited from CtInvocation class which can be seen in Figure 3.2. In the Code block 3.5, 5. code line is a Void Return EUnit. 3. and 4. code lines can also be as Void Return EUnit but they are already identified as Same Action Sequence EUnits, thus they aren't Void Return EUnits.

Same Action Sequence EUnits are the statements that uses methods but the parameters are different. In other words, it is the group of the same Void Return EUnits with different arguments. In Code block 3.5, 3. and 4. code lines are identified as Same Action Sequence EUnits.

Data Facilitator EUnits are the statements that changes the value of a parameter, variable or a field that is used in the other EUnits. In Code block 3.5, 2. and 7. code lines are Data Facilitator EUnits.

Controlling EUnits are the statements such as; if, while, do-while, for, switch and have control on the other EUnits. In other words, if an EUnit is in the some if

statement, that if statement will be stated as Controlling EUnit of the related EUnit. In Code block 3.5, 6. code line is a Controlling EUnit.

3.2.2 Extraction of EUnits

In the following subsections, how Autocomment extracts the EUnits from Java source code will be explained in depth. Extracted EUnits will later be used in comment generation as they will be used to inform developer about the important statements which are critical to know before further changing the related method.

3.2.2.1 Extracting the Ending EUnits

Ending EUnits are the flow breaker statements in a method. They are one of the critical EUnits to inform developer that there is a statement which can end the method execution process. Sridhara et al observed that methods often perform a set of actions to accomplish a final action, which is the main purpose of the method[6]. In addition, Autocomment also considers the statements that throws exception or returns something as Ending EUnits and uses them in comment generation process.

3.2.2.2 Extracting the Void Return EUnits

Void Return EUnits are the statements that doesn't return or assign anything. The intuition is that when a method call doesn't return anything, it is called because of its side effects[6]. On the other hand, method calls that return a value will probably be used in building an important action.[6].

3.2.2.3 Extracting the Same Action Sequence EUnits

Same Action Sequence are the statements that has the same method call, does exactly same action but with a different parameter, variable or a field. Although

using a method call can decrease its importance rate as unique methods are intuitively has a bigger role on what the method does, they still have an effect and may contain critical information. Thus, they will be considered as only one statement and will be treated as one Void Return EUnit. One of the benefits of this to keep comments simple and short. Other benefit is that treating them as Void Return EUnits simplifies the comment generation process and require less code as they will be pushed down into same templates to create comments.

3.2.2.4 Extracting the Data Facilitator EUnits

Data Facilitator EUnits are the statements that changes the value of a parameter, variable or a field that is used in the other EUnits. Data Facilitator EUnits aren't used in comments directly but they rather be used in the selection of Controlling EUnits to create much more informative comments.

3.2.2.5 Extracting the Controlling EUnits

Controlling EUnits are the statements such as; if, while, do-while, for, switch and have control on the other EUnits. In order to be Controlling EUnits to be selected, the other EUnits must be selected first to choose the Controlling EUnits of them.

3.3 Comment Templates

Autocomment has 2 types of templates; summary templates and important statement templates.

Summary templates are used for summary comments which gives the developer a hint about what the method does. In order to select the template that suits the related method, a custom grammar is created using ANTLR4. ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading,

processing, executing, or translating structured text or binary files[19]. The grammar that is created using ANTLR4 will be used to choose the summary comment template.

Important statement templates are used for important statement comments which gives the developer a detailed info about what the method does. It contains critical statements which are converted to proper English sentences for developers to read.

3.3.1 Summary templates

The grammar that is being used in Autocomment with the rules is given in Code block 3.6 and the variables is given in Code blocks 3.7 to 3.11 to choose the best template.

We have created these templates by looking at the postag sequences in method names. By reviewing more than 5000 methods of open source projects, we have generalized the postag sequences by creating patterns which fits most of the postag sequences. As a result, summary templates are born.

In grammar, V is the variable for verbs, NPR is the variable for noun phrases but it's recursive so that it can have as much as postag next to it starting with NN like NNS, NN etc. , PP is the variable for prepositions, WS is for the white space used in rules and EOF is the end of rule indicator. Some of the variable have question mark next to them which means, that variable is optional, in other words, not required for rule to apply.

The process for Autocomment to use ANTLR to choose appropriate comment summary template goes as follows;

- Method name component is extracted from method signature
- Method name component is tokenized

- Method name component is pos-tagged
- Pos-tag sentence is sent to ANTLR grammar as an input
- ANTLR tries to choose which rule given in Code block 3.6 fits the given pos-tag sentence best
- Accepted rule is selected for comment generation process

In order to generate the given methodś comment, its postag sentence must fit one of the rules in the parser. For instance, for the method with tokenized name component of "create nodes" which has a postag sentence of "VB NNS" would apply to "one_verb_rule".

All of the summary comments starts with "This method" string and continues with depending on the effects that will be explained in the subsection 3.3.3.

```
1  // Defining grammar for comment titles
2  grammar CommentTitle;
3
4  one_verb_rule
5     : V NPR? EOF
6     ;
7
8  two_verb_rule
9     : V NPR v NPR? EOF
10    ;
11
12 first_prp_rule
13    : PP NPR? V? NPR? EOF
14    ;
```

Code Block 3.6: Grammar Rules

```
// Recursive noun phrase variable
NPR
  : 'NN'
  | 'NNS'
  | 'NNP'
  | 'NNPS'
  | 'JJ'
  | 'NN' ' ' NPR
  | 'NNS' ' ' NPR
  | 'NNP' ' ' NPR
  | 'NNPS' ' ' NPR
  | 'JJ' ' ' NPR
  ;
```

Code Block 3.7: NPR variable in grammar

```
// Recursive verb variable
VR
  : 'VB'
  | 'VBD'
  | 'VBG'
  | 'VBN'
  | 'VBP'
  | 'VBZ'
  | 'VB' ' ' VR
  | 'VBD' ' ' VR
  | 'VBG' ' ' VR
  | 'VBN' ' ' VR
  | 'VBP' ' ' VR
  | 'VBZ' ' ' VR
  ;
```

Code Block 3.8: VR Variable in grammar

```
// Non-recursive verb variable
V
 : 'VB'
 | 'VBD'
 | 'VBG'
 | 'VBN'
 | 'VBP'
 | 'VBZ'
 ;
```

Code Block 3.9: V Variable in grammar

```
// Preposition variable
PP
  : 'IN'
  ;
```

Code Block 3.10: PP Variable in grammar

```
// Skip spaces, tabs and new lines
WS
  : [ \t\r\n]+ -> skip
  ;
```

Code Block 3.11: WS Variable in grammar

3.3.2 Important statement templates

Important statement templates are used for important statement comments which gives the developer a detailed info about what the method does. Important statements are the comments that generated using EUnits. In other words, important statement comments are the EUnits which converted to English sentences for developers to read.

All important statement comment templates are given in Table 3.2

CtElement Type	Active Form	Passive Form
CtReturn	returns the [CtElement]	expression that is returned as [CtElement]
CtThrow	throws [CtElement]	exception that is thrown as [CtElement]
CtConstructorCall	creates new [CtElement]	new instance that is created as [CtElement]
CtAssignment	[assigned CtExpression] is assigned to [assignment CtExpression]	which [assigned CtExpression] is assigned by [assignment CtExpression]
CtVariableAccess	variable's type	-
CtLiteral	literal itself	-

Table 3.2: Important statement templates

In order to extract EUnits from source code, Autocomment utilizes a recursive approach using the AST of SPOON given in Figure 3.2. Almost for every class that is inherited from CtElement has a method convert the given CtElement to English sentence. In other words, they have a string representation to be used as important statement comment. It has to work recursively as CtElements may have other CtElements in their representation. For example, as given in Code block 3.12, a return statement may have a constructor call which also needs to be converted to English sentence.

```
1  // Returns new instance that is created as Student
2  return new Student();
```

Code Block 3.12: Return statement with comment

All the CtElements that have a string representation also have an option to that representation to be either active or passive sentence. When comment starts to be prepared, the first CtElement is always have an active form and all other CtElements will have passive form. For example in Code block 3.12, the comment has an active return statement comment and inside, there is a constructor call comment as its in passive form. So that, "new instance that is created as [CtElement].type" is Contstructor Call's passive form whereas "creates new [CtElement]" is its active form. "[CtElement]" is the related element's comment which will be used in

to get element's comment. In given Code block 3.12, given CtElement is CtVariable which doesn't have active or passive form but just returns element's type as string to be used in comments.

CtInvocation is different from other CtElements because it contains a method call in its body. Therefore, its comment can not be generated as the other CtElements. Generating comments using method signature has already done in summary comments in subsection 3.3.1, so using the same methodology, Autocomment can create comments for CtInvocation elements but without the prefix "this method". Another problem is that method calls in CtInvocation statements is not limited to one. For example, in the CtInvocation element like"student.getType().toString()" there are two method calls which are "getType()" and "toString()" which "getType()" is being the first method call and "toString()" is the second method call. That's why, it also needs to be recursive in itself. Autocomment approaches this problem as follows;

- If CtInvocation statement is not in another statement such as CtIf and it's the first method call, then use its active form
- If CtInvocation statement is not in another statement and it is not the first method call, then use its passive form
- Otherwise, use its passive form

Finally, if the EUnit element has a parent Controlling EUnit, then related element's passive string representation will be used in the Controlling EUnit element's active form sentence. In order to build the string representation for Controlling EUnit, its conditions have to be put together recursively as conditions can be more than just one condition. Autocomment approaches this problem as it gets the parent condition of Controlling EUnit element which contains all other conditions of related Controlling EUnit element, then extracts other conditions recursively until no conditions can be extracted any more. It does this by extracting both left and right operand of condition, then does the same thing for both

Condition Type	Template
NOT	[left_operand] is not true
NOT EQUAL TO	[left_operand] is not equal to [right_operand]
EQUAL TO	[left_operand] is equal to [right_operand]
AND	[left_operand] and [right_operand]
OR	[left_operand] or [right_operand]
GREATER THAN	[left_operand] is greater than [right_operand]
LESS THAN	[left_operand] is less than [right_operand]
GREATER or EQUAL TO	[left_operand] is greater than or equal to [right_operand]
LESS or EQUAL TO	[left_operand] is less than or equal to [right_operand]
INSTANCE OF	[left_operand] is instance of [right_operand]
ADDITION	[left_operand] plus [right_operand]
SUBTRACTION	[left_operand] minus [right_operand]
MULTIPLICATION	[left_operand] multiplied by [right_operand]
DIVISION	[left_operand] divided by [right_operand]
MOD	[left_operand] mod [right_operand]

Table 3.3: Important statement condition templates

left and right condition in their respective operands. In Table 3.3 all condition templates for important statements is given.

3.3.3 Effects on comments

In this subsection, effects that changes the templates will be explained in detail. These changes affects summary comments and CtInvocation comments.

3.3.3.1 POS tags

As given in Figure Code block 3.6 postags are primary reason to change template. There are 3 rules which utilize postags and its order; one verb rule, two verb rule and first preposition rule.

The flow of generating summary comments except for boolean methods is given in Figure 3.7, for only boolean methods is given in Figure 3.9 and for CtInvocation statement's string representation templates which will be used in important

statement comments is given in Figure 3.8. All of this processes will be explained in detail in this section.

Firstly, regardless of summary comment template, Autocomment starts active summary comment with the string “This method”, then depending on rules, summary comment is got appended different string combinations.

For one verb rule, Autocomment firstly checks either the first verb has the postag “VBZ” or “VB”. If it has the first verb with postag “VBZ” and has parameters, summary comment is appended with parameters string which will be explained in detail in subsection 3.3.3.3. Then, the verb and noun phrase is appended respectively to summary comment. Finally, a comment phrase is prepared for parameters is appended to summary comment. This comment will be used as summary type comment for related method. This is an active form of comment. This comment’s passive form will be used for CtInvocation of important statement comments as most of the process is same but the combination of strings and verbs tenses differs. For the passive form of one verb rule with “VBZ” postagged verb, it firstly gets the CtInvocation method with its all of the child CtInvocation statements. For example, “object.getType().toString()” is a statement which has 2 CtInvocation statements in it which are “getType()” and “toString()” and has the target “object”. As in the example, all CtInvocation statements go for the same preprocesses which is mentioned in section 3.1. Then, if it is the first CtInvocation statement, its verb is appended to important statement comment with its noun phrase, moreover if it isn’t first CtInvocation statement and if it has noun phrase, its noun phrase and verb with 3rd form is appended to important statement comment respectively, if doesn’t have a noun phrase, then only verb with 3rd form and appropriate preposition of related verb is appended to comment. This processes for important statement comment for CtInvocation statement continues until the last CtInvocation statement of the current code line is found. After that, if there is a target object, its appended to comment, if not, then “this instance” string is appended.

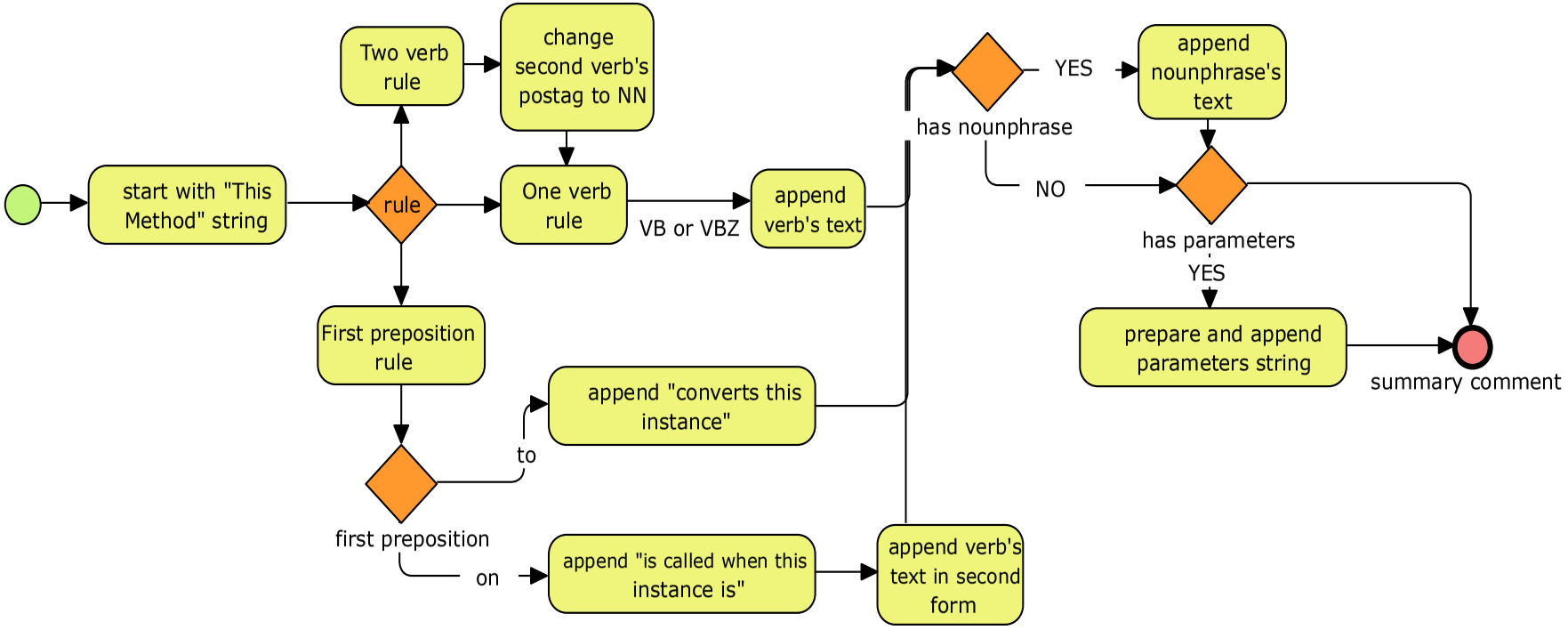


Figure 3.7: Summary templates for all methods except boolean methods

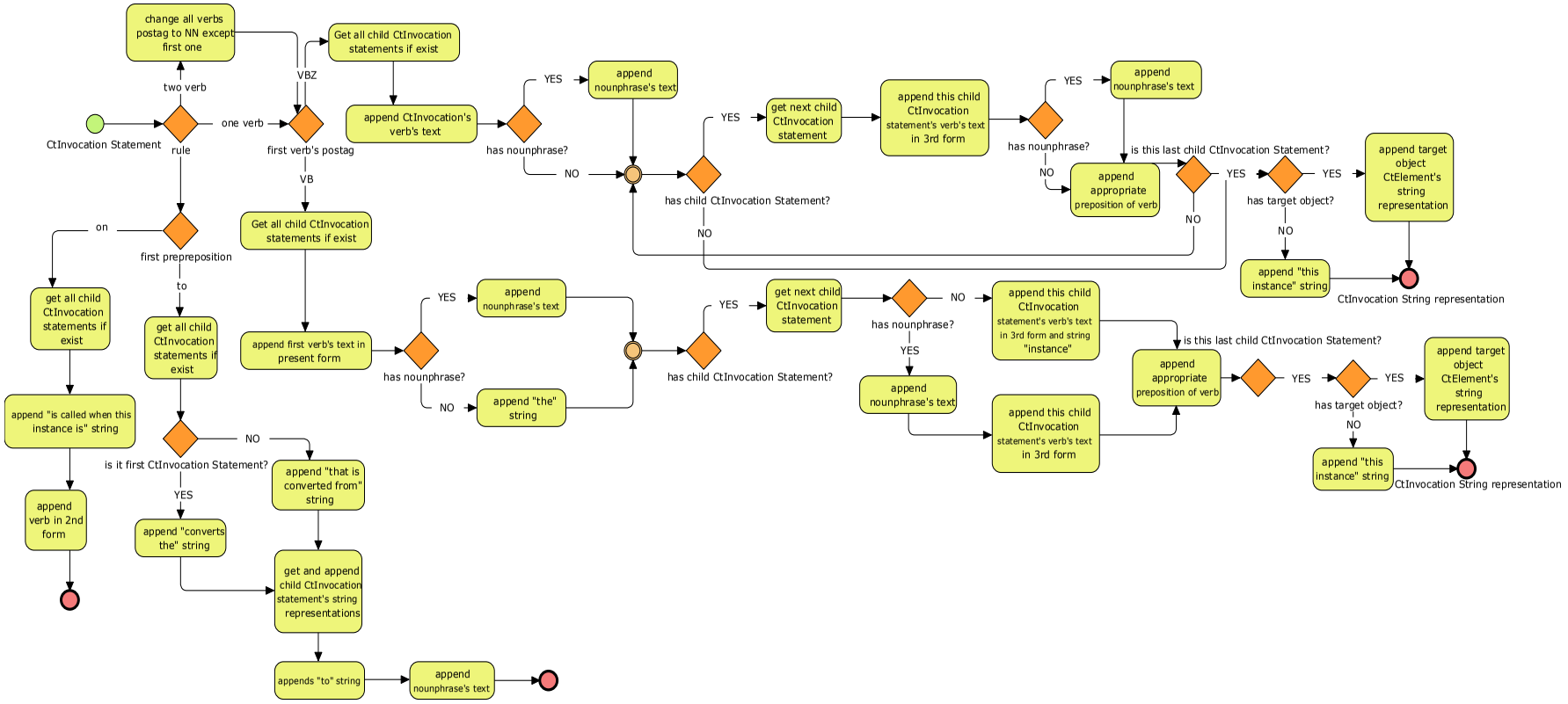


Figure 3.8: Templates for CtInvocation statement's string representations

For one verb rule with the postagged verb of "VB", comment starts with "This method" string again on its active form. It continues with appending verb in present tense and noun phrase if exists. Finally prepares and appends string for parameters. For passive form for CtInvocation statements, it appends the first verb in its present form and noun phrase if it exists, if noun phrase doesn't exist, then it appends the string "the". After the first CtInvocation statement, it appends noun phrase firstly, then verb in its third form and appropriate preposition of verb. If the noun phrase doesn't exist, then it appends verb in third form, then the string "instance" and appropriate preposition of verb. This process continues until the last CtInvocation statement on that code line. Finally, for the final CtInvocation statement, if target exists, it appends the target object, if not, then it appends the string "this instance".

Two verb rule is actually has the same process and strings for appending as Autocomment expects only one verb in method signature. In [20], there is a saying that "It is often tempting to create functions that have multiple sections that perform a series of operations. Functions of this kind do more than one thing, and should be converted into many smaller functions, each of which does one thing.". It also says "Choose names that make the workings of a function or variable unambiguous.". One can conclude that using only one verb would increase readability which is one of the necessary steps towards automatic comment generation ambiguous method names.

For first preposition rule, it firstly appends the string "This method" for active form. If the first preposition is "to", then appends "converts this instance" and if noun phrase exists, it appends "to", the noun phrase respectively, if not, then it only prepares and appends the string for parameters. If the first preposition is "on", it appends the string "is called when this instance is" and verb in second form. For the CtInvocation statement comment, if it is the first CtInvocation statement and the first preposition is "to" then it appends "converts the". If it also has noun phrase and has more CtInvocation statements in the current code line, it gets the comments that is generated for those CtInvocation statements

and appends those comments, then appends the preposition “to” and noun phrase respectively. This part is recursive in the middle of comment since Autocomment has to know what this method converts to. If the first preposition is “on”, then it appends “is called when this instance is” and verb in second form respectively. If it is the first CtInvocation statement and has noun phrase, it appends the noun phrase and the string “that is converted from”, if not, then appends “to”. Then, if it is the last CtInvocation statement and has target, it appends the target object, if it doesn’t have a target, then it appends the string “this instance”.

Autocomment also considers all other words with the postags that doesn’t exist in rules such as “RB”, “JJ” etc. as “NN” in order to keep using one of rules. By doing so, all method names become valid and has a summary comment template which can be used in summary comment generation process.

In Code block 3.13, an auto-generated comment for a boolean method with NN postag is given. This method has the postag sequence of “VBZ NN” for the words “is online”.

```
1  // This method checks whether this instance is online or not
2  boolean isOnline() { ... }
```

Code Block 3.13: Generated comment for a boolean method with NN postag

In Code block 3.14, an auto-generated comment for a boolean method with “JJ” postag is also given. This method has the postag sequence of “VBZ JJ” for the words “is dynamic”. Replacing the postag “JJ” with “NN” for the “isDynamic()” method makes it possible to use the same comment template. This example shows that replacing “JJ” postag with “NN” makes the comment templates usable for postags which are not in comment template grammar without any information loss in comments.

```
1  // This method checks whether this instance is dynamic or not
2  boolean isDynamic() { ... }
```

Code Block 3.14: Generated comment for a boolean method with JJ postag

3.3.3.2 Method return types

Autocomment approaches method types whether it's boolean or not. Another approach would be whether the method has a return type or not but since Autocomment puts important statements such as return statement in important statement comments, there is no need to put return statement info to summary comment as well. Summary comment generation process for boolean methods is given in Figure 3.9 and will be explained in this subsection.

For methods with the return type of boolean, summary comment starts as "This method checks whether the" string and then, there are two options to choose from; if the method has verb with postag "VB", then verb with "ing" suffix and then, if it doesn't have parameters "this instance" string is appended. Finally, if it has noun phrase, it is appended, then string for parameters and "is successful or not" string is appended. If the verb has the postag "VBZ", then the same process as "VB" verb applies but verb doesn't get "ing" suffix and the end string is just "or not" rather than "successful or not".

Return types doesn't affect the comment generation process for important statements therefore whether boolean or not, important statements would all be the same for all methods.

Two verb rule and first preposition is not supported for boolean methods due to boolean methods' naming convention. Boolean methods should start with a verb rather than preposition which means that developer should change method's name with a one that applies convention rules. One solution for this problem would be recommending a better name to developers but it is beyond the scope of the current Autocomment right now and will be further implemented in the future.

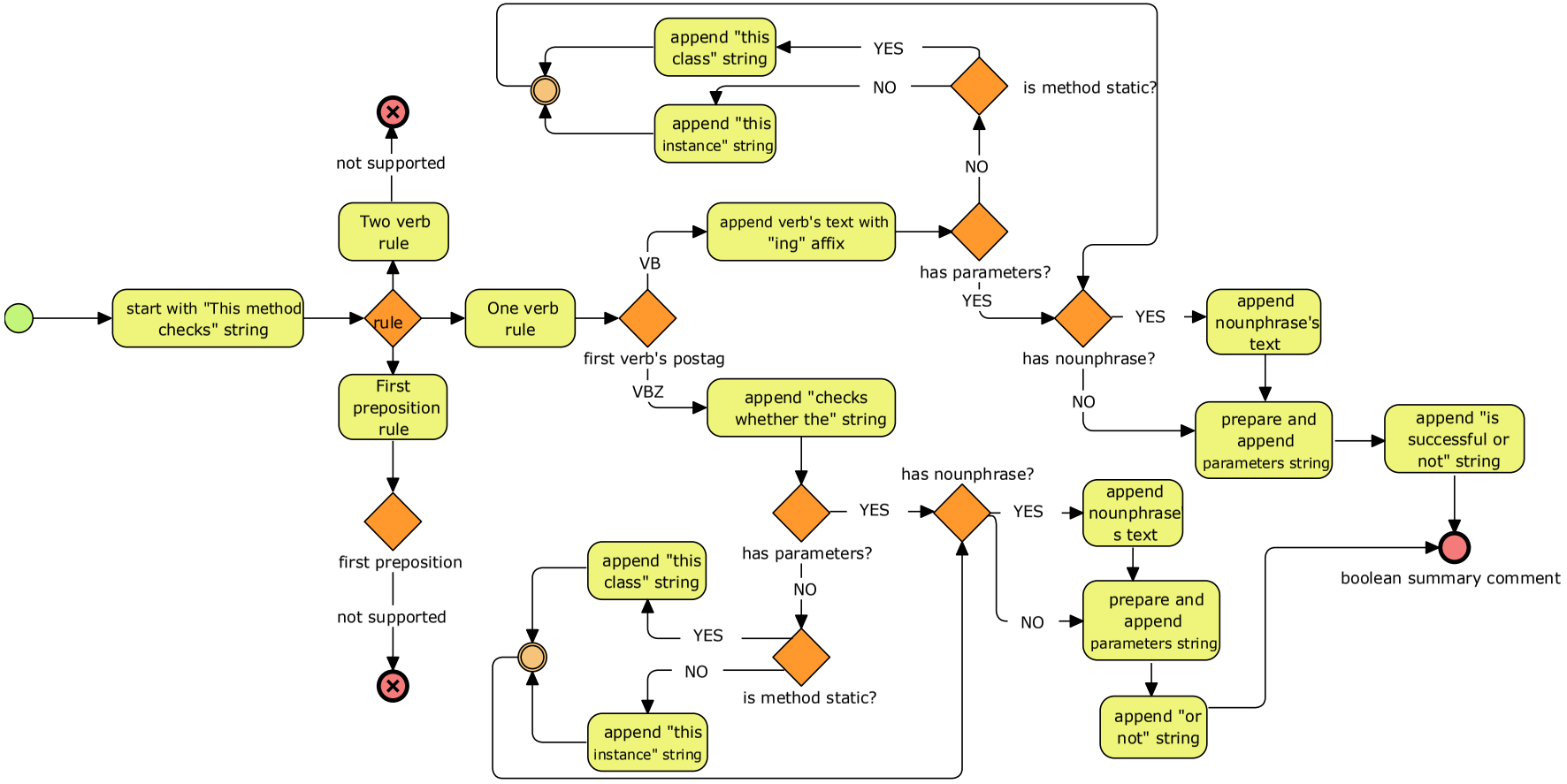


Figure 3.9: Summary template for boolean methods

3.3.3.3 Parameters

In summary templates, it is mentioned that a string for parameters is prepared and appended to summary comment. In this subsection, that string will be explained. In Figure 3.10, parameter string generation process is given.

Firstly, Autocomment checks whether there are parameters or not. If there isn't any parameters, then it checks whether the method is static or not. If it is static, then only "this class" string will be returned and appended to summary. If it isn't static, then only "this instance" will be returned and appended to summary. If there are parameters, Autocomment firstly gets the appropriate preposition for the given verb. For example, if the verb is "add" then the preposition for the verb would be "to" and if the verb is "find", then the preposition would be "from". If the given verb is not in Autocomment's list of verbs, then default preposition which is "using" will be used. After that, it appends the string "the given" and the parameters one by one. If there are multiple parameters, then those parameters will be separated by the string "and".

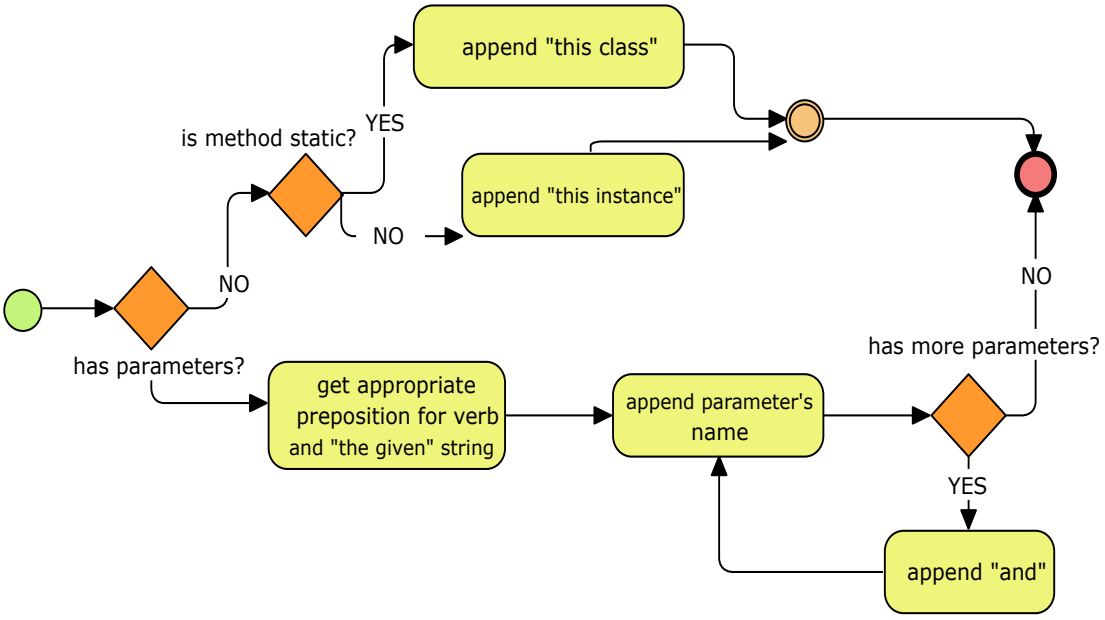


Figure 3.10: Parameter string generation process

3.4 Comment Examples

In this section, comments which are generated by Autocomment are given with their source code.

```
1  /**
2   * Summary: This method gets search from this class
3   *
4   * Important Statements:
5   * 1-) if isConfigured is not true, then throws Exception
6   * 2-) if searchHandler is not null, then throws StackOverflowError
7   * 3-) returns the new instance that is created as Search using the search
8   * Handler
9   */
10 public static Search getSearch(){
11    SearchHandler searchHandler = getSearchHandler();
12    if (!(isConfigured))
13       throw new Exception(("This JDisc does not have 'search' " +
    "configured."));
14
15    if (!(searchHandler == null))
16       throw new StackOverflowError();
17
18    return new Search(searchHandler);
19 }
```

Code Block 3.15: get_search method with comments

```java
/**
 * Summary: This method converts this instance to xml string using the given
 * object
 *
 * Important Statements:
 * 1-) returns the xml that is converted from xstream
 */
public String toXMLString(Object object){
   return xstream.toXML(object);
}
```

Code Block 3.16: toXMLString method with comments

```java
/**
 * Summary: This method finds best match from the given versionSpec and
 * versions
 *
 * Important Statements:
 * 1-) returns the bestMatch
 */
protected static Version findBestMatch(VersionSpecification versionSpec,
 Set<Version> versions){
   Version bestMatch = null;
   for (Version version : versions){
      if((version == null) || (!(versionSpec.matches(version))))
         continue;
      if((bestMatch == null) || ((bestMatch.compareTo(version)) < 0))
         bestMatch = version;
   }
   return bestMatch;
}
```

Code Block 3.17: findBestMatch method with comments

```
1  /**
2   * Summary: This method registers using the given id and component
3   *
4   * Important Statements:
5   * 1-) if frozen , then throws IllegalStateException
6   * 2-) if componentVersionsByName is null , then put instance into
7   * componentsByNameByNamespace
8   * 3-) put instance into componentVersions
9   * 4-) put instance into componentsById
10  * 5-) if componentVersions is null , then put instance into
11  * componentVersionsByName
12  */
13 public void register(ComponentId id, COMPONENT component){
14    if(frozen)
15       throw new IllegalStateException("Cannot modify a frozen component
    registry");
16    Map<String, Map<Version, COMPONENT>> componentVersionsByName =
    componentsByNameByNamespace.get(id.getNamespace());
17    if(componentVersionsByName == nul){
18       componentVersionsByName = new LinkedHashMap();
19       componentsByNameByNamespace.put(id.getNamespace(),
    componentVersionsByName);
20    }
21    Map<Version, COMPONENT> componentVersions =
    componentVersionsByName.get(id.getName());
22    if(componentVersions == nul){
23       componentVersions = new LinkedHashMap();
24       componentsByNameByName.put(id.getName(), componentVersions);
25    }
26    componentVersions.put(id.getVersion(), component);
27    componentsById.put(id, component);
28 }
```

Code Block 3.18: register method with comments

```
1  /**
2   * Summary: This method checks whether deleteing the given page is
    * successful or not.
3   *
4   * Important Statements:
5   * 1-) if key deleteed using configKeys is equal to E_OK , then infoed
6   * instance using logger
7   * 2-) unless page is deleteed equal to E_OK , returns the E_ERROR
8   * 3-) returns the E_OK
9   */
10  public boolean delete(Page page) {
11      if ((deletePage(page)) == (E_OK)) {
12          if ((registry.deleteReference(page.name)) == (E_OK)) {
13              if ((configKeys.deleteKey(page.name)) == (E_OK)) {
14                  logger.info("page deleted");
15              }else {
16                  logger.info("configKey not deleted");
17              }
18          }else {
19              logger.info("deleteReference from registry failed");
20          }
21      }else {
22          logger.info("delete failed");
23          return E_ERROR;
24      }
25      return E_OK;
26  }
```

Code Block 3.19: delete method with comments

```
1   /**
2    * Summary: This method is called when this instance is createed
3    *
4    * Important Statements:
5    * 1-) calls when super is created
6    * 2-) gets int from the mPrefs is assigned to mCurViewMode
7    */
8   protected void onCreate(Bundle savedInstanceState){
9      super.onCreate(savedInstanceState);
10     SharedPreferences mPrefs = getSharedPreferences();
11     mCurViewMode = mPrefs.getInt("view-mode", DAY_VIEW_MODE);
12  }
```

Code Block 3.20: oncreate method with comments [21]

Chapter 4

Evaluation

In order to find out how viable the Autocomment's auto-generated comments are, a survey with 50 participants with 10 questions has been done. There wasn't any time limit to answer all questions and all questions were required to be answered to finish the survey. Participants have finished the survey in 3 minutes and 17 seconds on average. Participants who have finished the survey less than 90 seconds(6 participants) are excluded from the results as the intuition is that those participants haven't filled the survey carefully. Participants haven't be informed in anyway except the survey description which says "Autocomment is a tool to generate comments for methods only using source code. This survey results will be used in my master thesis. Thank you for participating. -Eren YILDIZ" to keep the participants unbiased and to get the most accurate feedback from them. Survey has been done in a survey website SurveyMonkey[22].

First question was the work experience in programming in months. In the result given in Figure 4.1, average work experience of 50 participants is 47 months which is almost 4 years. This means the survey answers and the results are dependable.

Second question was given in Figure 4.2. The question "Do you do code review? If so, then do you think that comments are helpful for code review?" was asked to participants.

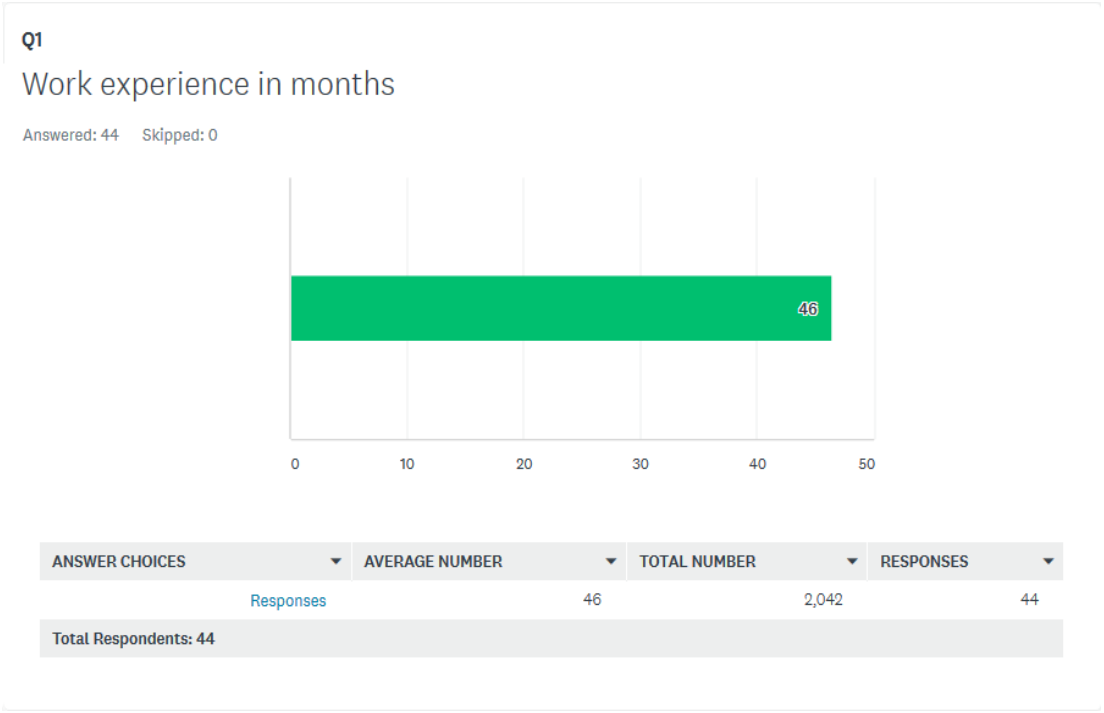


Figure 4.1: Result of the first question of survey

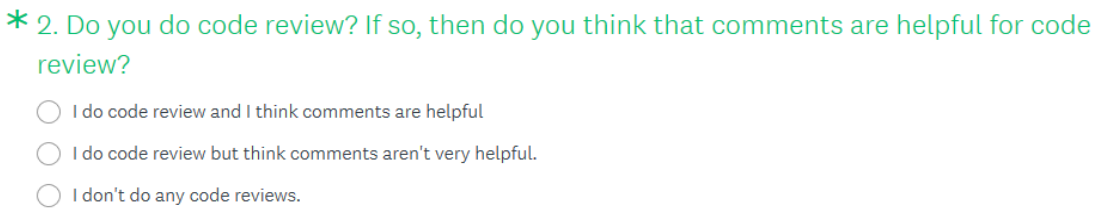
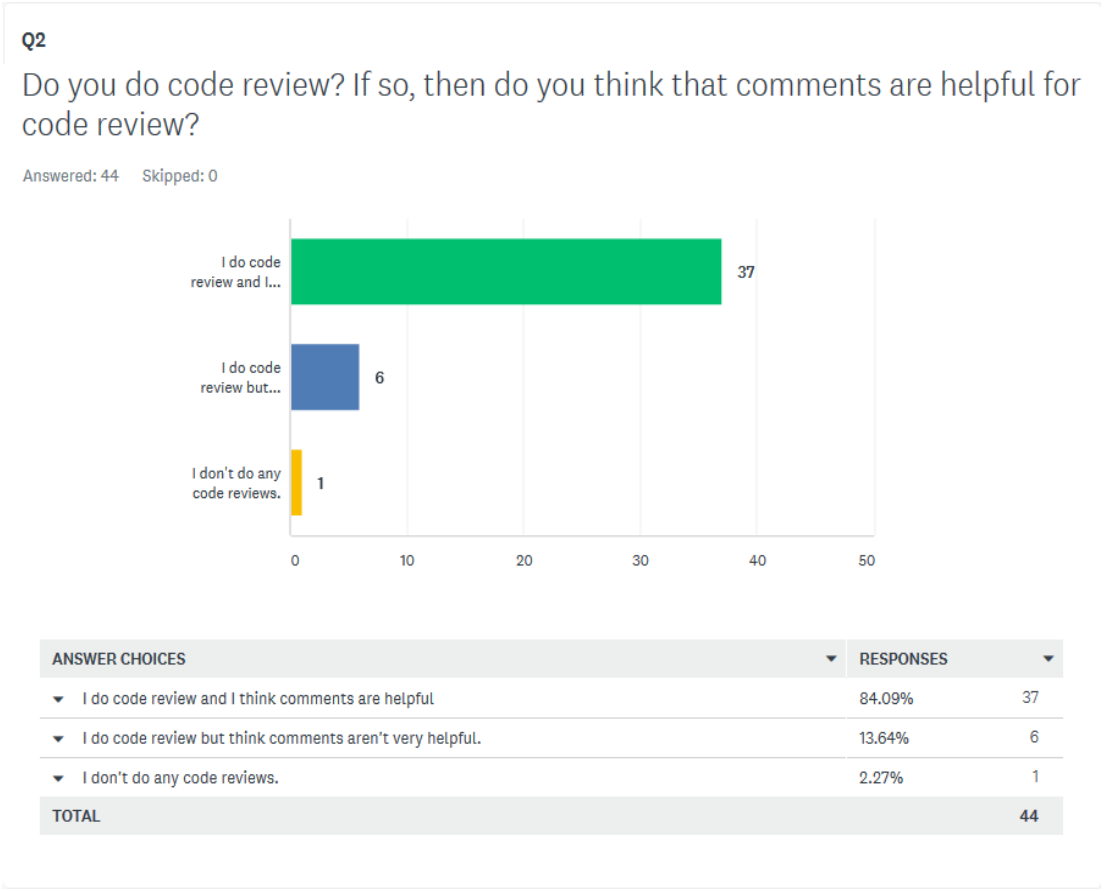


Figure 4.2: Second question of survey

As in Figure 4.3, results says that 41 out of 50 people in programming industry do code review and find comments helpful. This implies that comments are important and needed. This also justify the Autocomment's aims and objectives as it stated in the section 1.3. It was also asked to 8 people who do code reviews but why don't find the comments helpful to get a better feedback. Most of them stated that they actually like good quality comments but most of the developers write it poorly. They also says that sometimes it is better not to have any comments rather than ambiguous comments which lead the other developers who follow the related project write buggy codes unintentionally. This means that they actually find comments helpful but they need to be descriptive and unambiguous.



ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	84.09%	37
I do code review but think comments aren't very helpful.	13.64%	6
I don't do any code reviews.	2.27%	1
TOTAL		44

Figure 4.3: Result of the second question of survey

Third question was given in Figure 4.4. The question "Which areas you have experience on?" was asked to participants. Question was multiple-choice which means that a participant can choose more than one option. For example, a participant can be tester, developer and researcher at the same time and choose accordingly. This question is asked to find out whether the comments are only helpful or not for the developers who write comment.

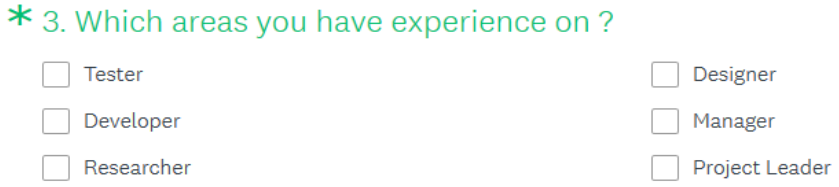


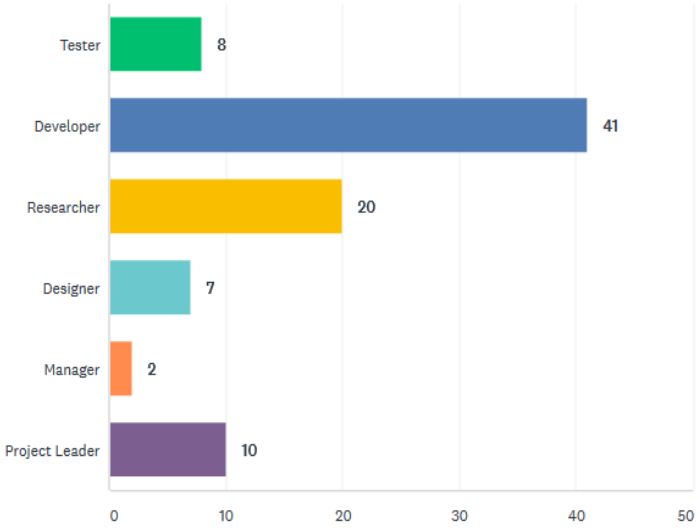
Figure 4.4: Third question of survey

As in Figure 4.5, people with wide range of roles have participated.

Q3

Which areas you have experience on ?

Answered: 44 Skipped: 0



ANSWER CHOICES	RESPONSES	
Tester	18.18%	8
Developer	93.18%	41
Researcher	45.45%	20
Designer	15.91%	7
Manager	4.55%	2
Project Leader	22.73%	10
Total Respondents: 44		

Figure 4.5: Result of the third question of survey

Questions from fourth to eighth were the rating questions of comments quality. All comment rating questions are given below in Figures 4.6, 4.8, 4.10, 4.12, 4.14 and the results are given in Figures 4.9, 4.11, 4.13, 4.15 respectively. Also in ninth question overall summary comments' quality and in tenth question, overall important statements comments' quality were asked and results are given in Figures 4.16, 4.17 respectively.

```java
/**
 * Summary: This method  gets search from this class
 *
 * Important Statements:
 * 1-) if isConfigured , then throws Exception
 * 2-) if searchHandler is not null , then throws StackOverflowError
 * 3-) returns the new instance that is created as Search using the searchHandler
 */
public static Search getSearch() {
    SearchHandler searchHandler = getSearchHandler();
    if (!(isConfigured))
        throw new Exception(("This JDisc does not have 'search' " + "configured."));

    if (!(searchHandler == null))
        throw new StackOverflowError();

    return new Search(searchHandler);
}
```

4.

Please rate this comment.

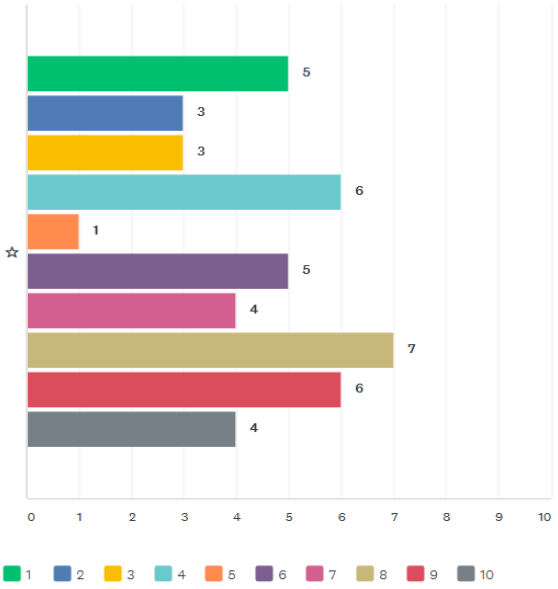
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

Figure 4.6: Fourth question of survey

Q4

Please rate this comment.

Answered: 44 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.36% 5	6.82% 3	6.82% 3	13.64% 6	2.27% 1	11.36% 5	9.09% 4	15.91% 7	13.64% 6	9.09% 4	44	5.84

Figure 4.7: Result of the fourth question of survey

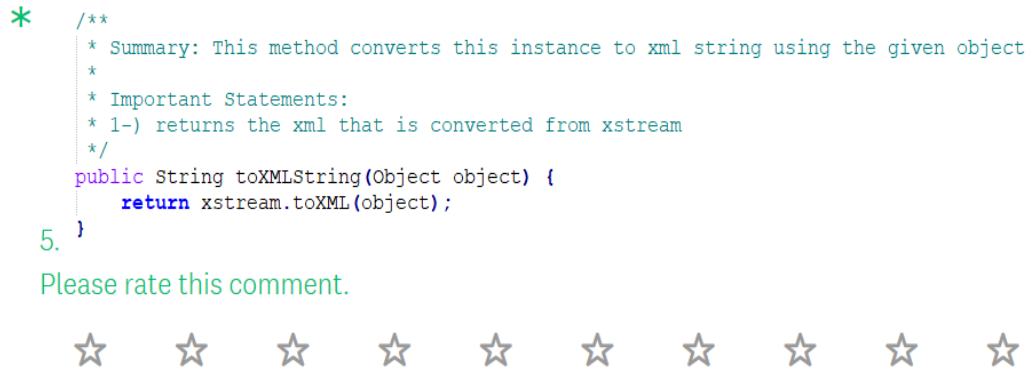
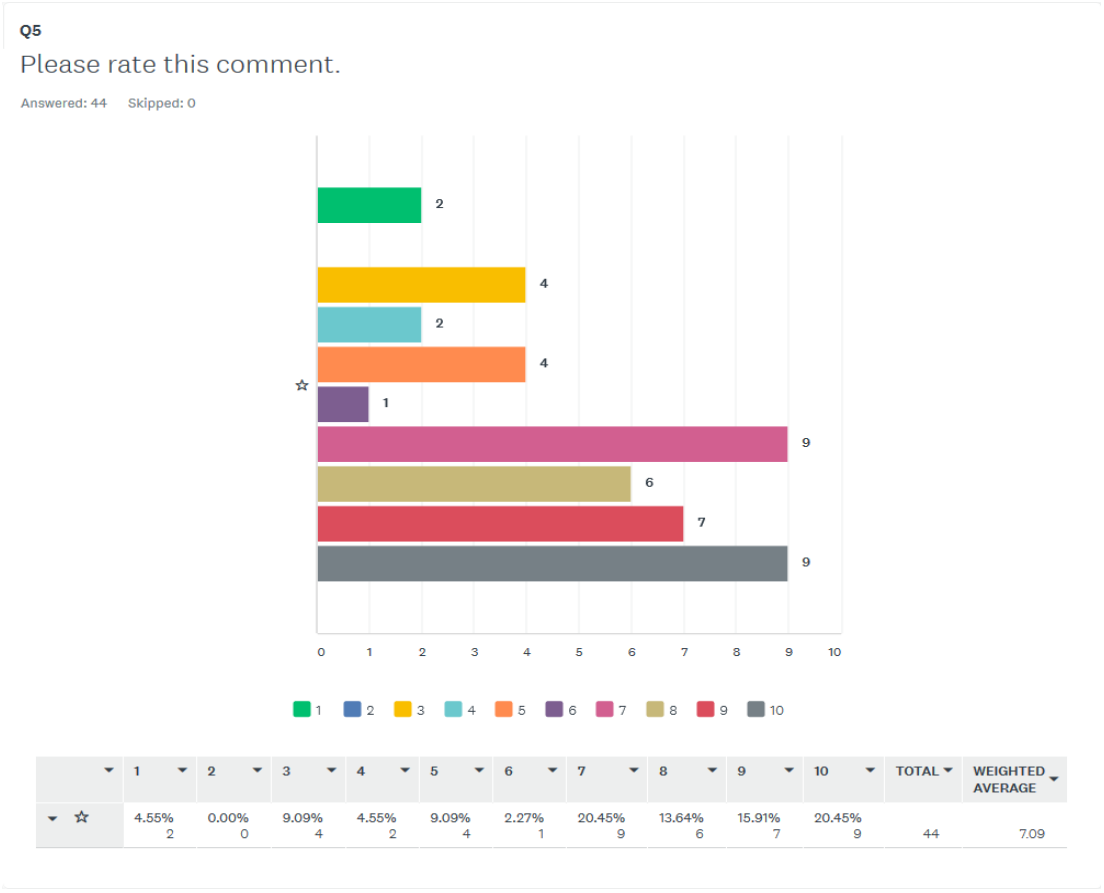


Figure 4.8: Fifth question of survey



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.55% 2	0.00% 0	9.09% 4	4.55% 2	9.09% 4	2.27% 1	20.45% 9	13.64% 6	15.91% 7	20.45% 9	44	7.09

Figure 4.9: Result of the fifth question of survey

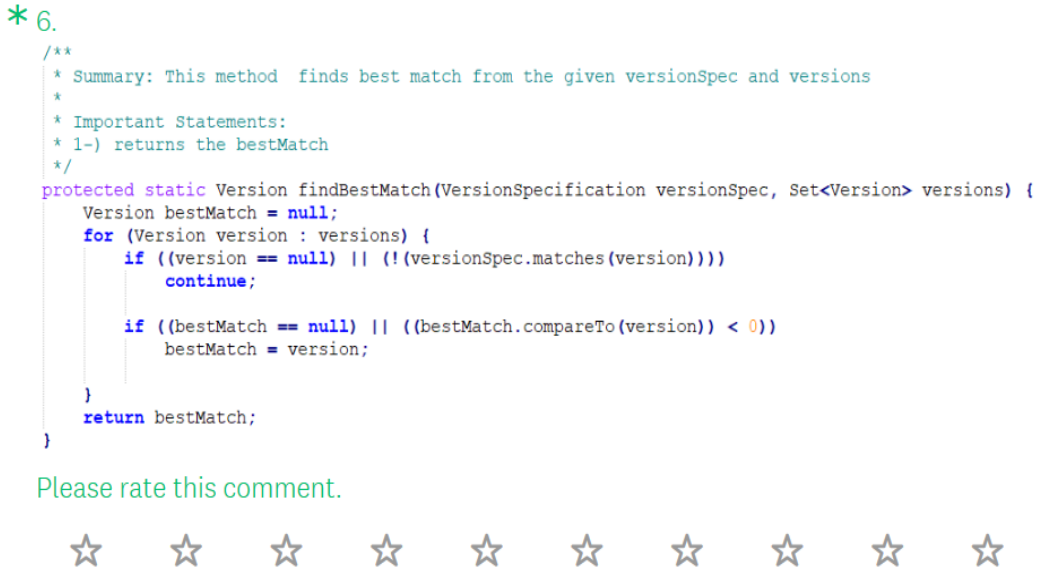


Figure 4.10: Sixth question of survey

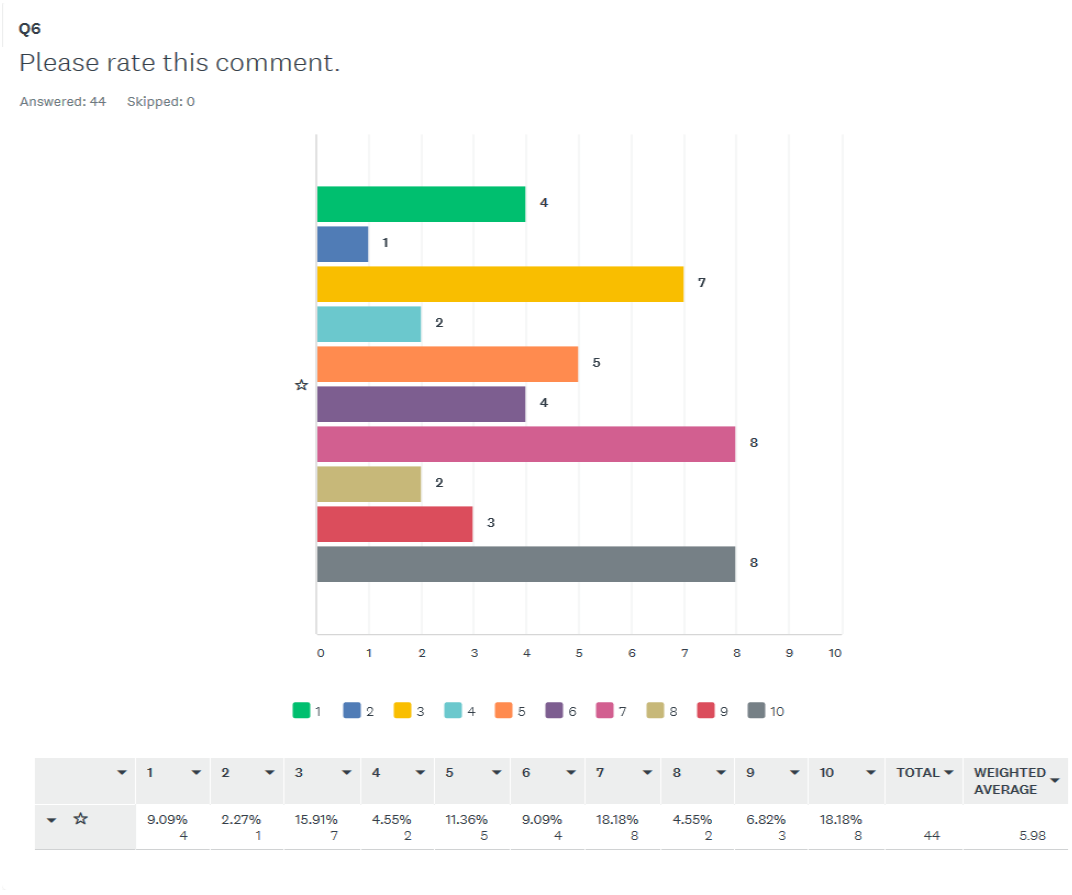


Figure 4.11: Result of the sixth question of survey

*7.

```
/**
 * Summary: This method  registers using the given id and component
 *
 * Important Statements:
 * 1-) if frozen , then throws IllegalStateException
 * 2-) if componentVersionsByName is null , then put instance into componentsByNameByNamespace
 * 3-) put instance into componentVersions
 * 4-) put instance into componentsById
 * 5-) if componentVersions is null , then put instance into componentVersionsByName
 */
public void register(ComponentId id, COMPONENT component) {
    if (frozen)
        throw new IllegalStateException("Cannot modify a frozen component registry");

    Map<String, Map<Version, COMPONENT>> componentVersionsByName = componentsByNameByNamespace.get(id.getNamespace());
    if (componentVersionsByName == null) {
        componentVersionsByName = new LinkedHashMap();
        componentsByNameByNamespace.put(id.getNamespace(), componentVersionsByName);
    }
    Map<Version, COMPONENT> componentVersions = componentVersionsByName.get(id.getName());
    if (componentVersions == null) {
        componentVersions = new LinkedHashMap();
        componentVersionsByName.put(id.getName(), componentVersions);
    }
    componentVersions.put(id.getVersion(), component);
    componentsById.put(id, component);
}
```

Please rate this comment.

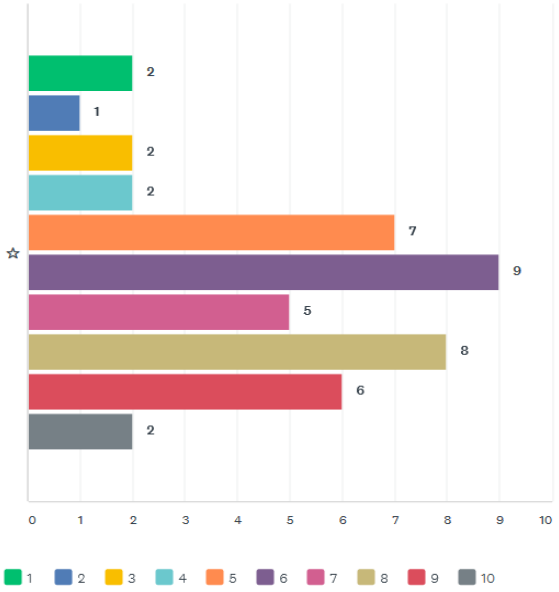
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

Figure 4.12: Seventh question of survey

Q7

Please rate this comment.

Answered: 44 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.55% 2	2.27% 1	4.55% 2	4.55% 2	15.91% 7	20.45% 9	11.36% 5	18.18% 8	13.64% 6	4.55% 2	44	6.36

Figure 4.13: Result of the seventh question of survey

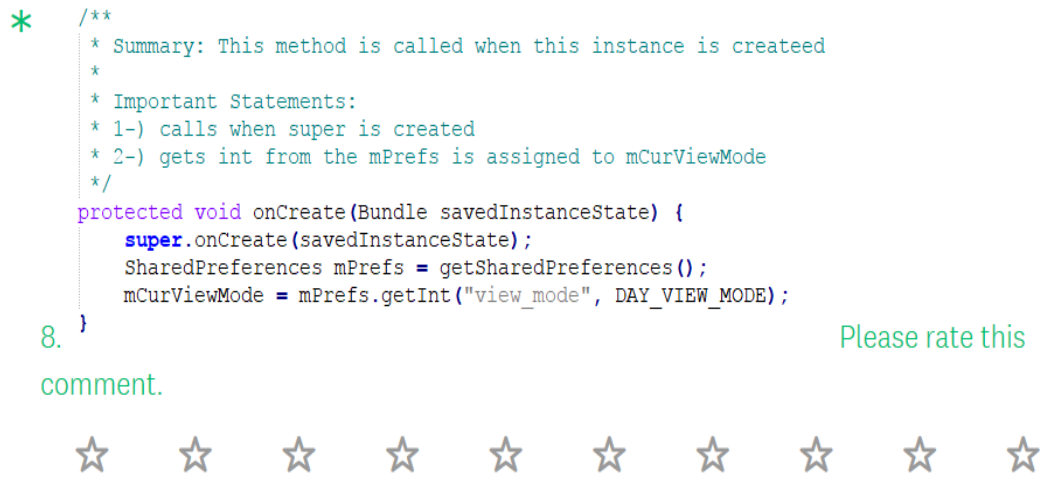
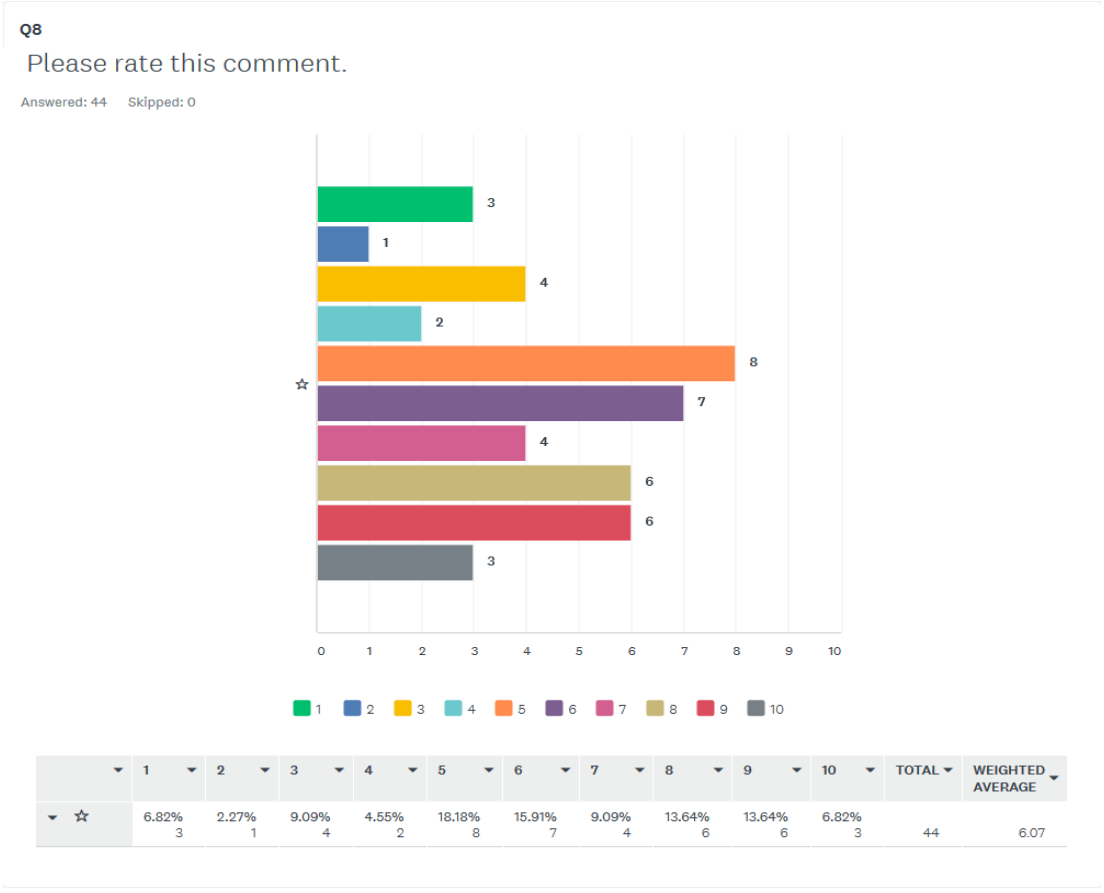


Figure 4.14: Eighth question of survey



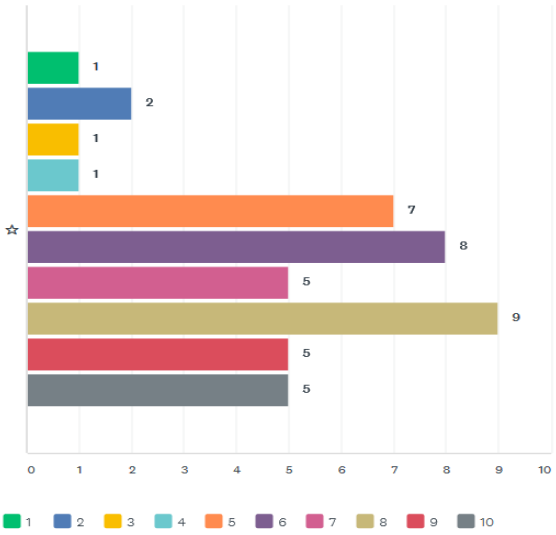
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	6.82% 3	2.27% 1	9.09% 4	4.55% 2	18.18% 8	15.91% 7	9.09% 4	13.64% 6	13.64% 6	6.82% 3	44	6.07

Figure 4.15: Result of the eighth question of survey

Q9

Please rate the overall quality of summary comments.

Answered: 44 Skipped: 0



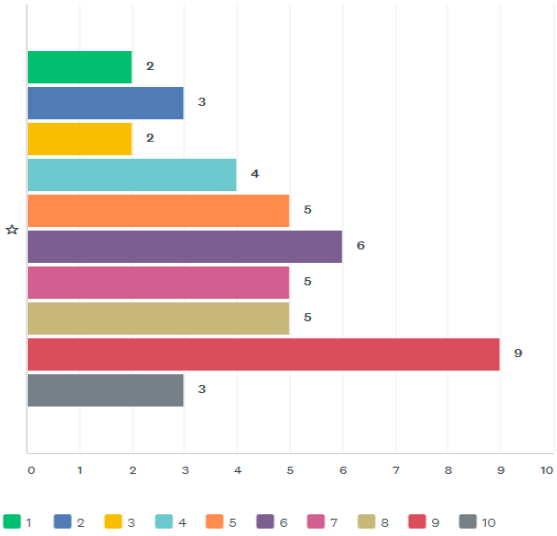
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	2.27% 1	4.55% 2	2.27% 1	2.27% 1	15.91% 7	18.18% 8	11.36% 5	20.45% 9	11.36% 5	11.36% 5	44	6.75

Figure 4.16: Result of the ninth question of survey

Q10

Please rate the overall quality of important statement comments.

Answered: 44 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.55% 2	6.82% 3	4.55% 2	9.09% 4	11.36% 5	13.64% 6	11.36% 5	11.36% 5	20.45% 9	6.82% 3	44	6.30

Figure 4.17: Result of the tenth question of survey

Results say that, with average rating of 6.58 for summary comments and 6.14 for important statement comments, people actually care about comments and want them in the projects. Although there isn't any question about "What can be done to improve Autocomment?" or "Why did you rate low for that question specifically or overall?" but some of them sent their feedback regardless. One of the participant says that summary comments are not necessary as they give very little information about how the code does its work. When it is explained to the participant that the purpose of the summary comments is to explained to reader "what" the method does instead of "how" method does it. Moreover, summary comments designed to be straightforward to be readable for non-developers as well. After the explanation, participant agreed and find summary comments acceptable and appropriate for its purpose. Another participant asked that "Why comment format does not look like javadoc?". Participant wanted the comments to be in javadoc format yet Autocomment's target audience doesn't limited to developers but it is much more broader than that. Thus, it needs a new and different format than javadoc. When it is explained, participant loved the idea of having a comment that also can be readable by non-developers such as customers. Another participant asked "What does the statement mean?". Like all other answers to feedbacks, the answer was not told to the participant before finishing the survey. After participant has finished it, it is explained that statement actually means "a code line". Participant says that "code line" is more explanatory than just "statement". Participant's feedback is noted to improve Autocomment's comment quality. Another participant asked "Why there isn't any information about object like its class rather than just its name?". Autocomment's comments are designed to be simple but informative as possible. In this regard, only the variable name's are used in comments yet sometimes, as in the Figure 4.14, variable names don't really provide any useful information about its context like variable "mCurViewMode". This is due to the fact that not all developers follow the clean code rules as in the book [20] so in order the participant's request to be included in Autocomment, there needs to be another research about variable names informativeness so that if it is informative enough, it can be used

in comment, if not, then its class name also needs to be included in comment. In addition, participant says that, whether the variable name is informative or not, there needs to be an option to toggle class names on and off as reader may want to look at the class name regardless due to the polymorphism. This feature request is noted to be researched and added to Autocomment in the future. Some of the participants reasoned that they gave low rating due to typos. Autocomment depends on Stanford's NLP toolkit heavily to overcome problems like this. As in the Figure 4.14, eighth question has typo in the word "createed". Reason behind thist is that Stanford's NLP toolkit doesn't have a feature to change form of given verb. Therefore Autocomment uses a local dictionary to change word's form but it's capacity isn't enough. One of the Stanford's NLP toolkit's developer say that this feature will be available in the future release so when this feature is available, Autocomment will not have typos any more. Another participant, who is a project leader, researcher and also developer said that "Autocomment is a lovely project which does the comment writing for me. I like comments as it let me follow the developers who work under me yet developers are tend to forget to write comment often, or they are just being lazy. Autocomment gives those people opportunity to write comments as with Autocomment, they don't have to write comment from scratch, but if necessary, they can just edit or add new information about the context into comment." This is one of the aims of Autocomment and getting this kind of feedback from a project leader makes it viable.

One important thing to note that, none of the participants find the comments untrustworthy. In fact, some of the participant stated that Autocomment's comments were able to captured critical and only the critical statements. This proves that Autocomment is reliable.

Results are also grouped by their roles and given in appendix.

Chapter 5

Conclusion

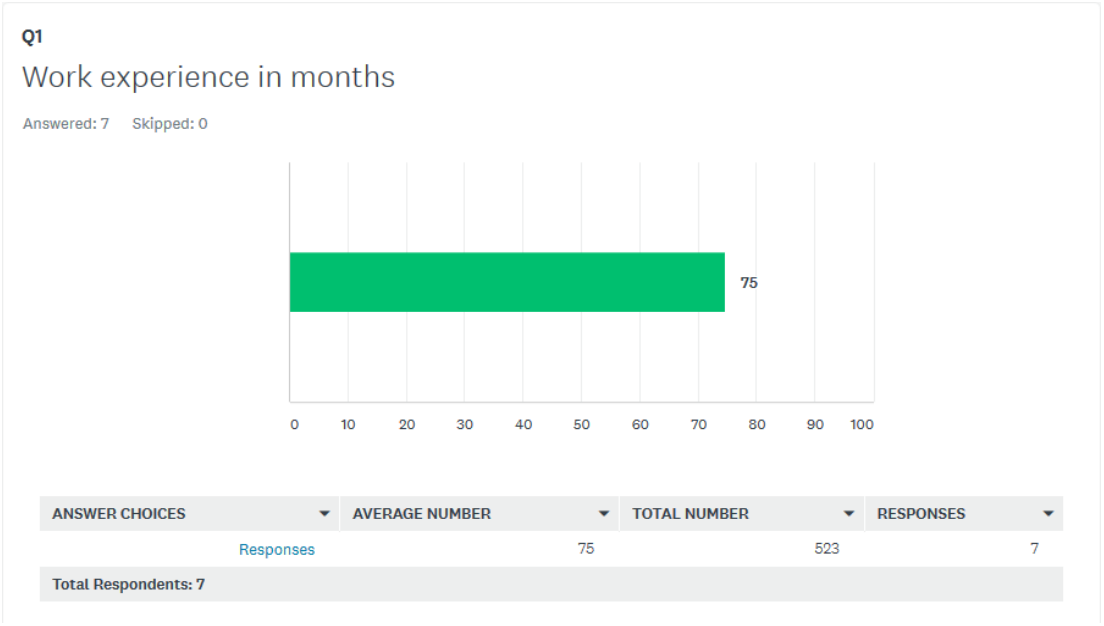
In this paper, an automatic comment generation framework named Autocomment is presented. Autocomment is able to generate 2 types of comments; summary comments which answers the question of "what does this method do?" and important statement comments which answers the question of "how does this method accomplish its tasks?". By utilizing the AST and using a rule based approach, Autocomment is able to generate those comments automatically.

As the survey results imply, Autocomment is able to find the most important statements in the method and create reliable comments for readers including developers, managers, testers, designers, project leaders, researchers and the people who are working in programming industry.

For future works, Autocomment can be extended with many other features such as; ambiguous variable name detection, method and variable name recommendation, use of call graphs to enrich the contextual information of comments etc.

Appendix

Survey Results of Designers



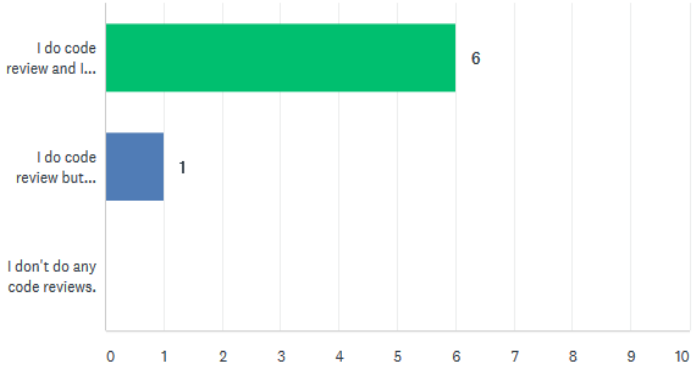
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	75	523	7
Total Respondents: 7			

Result of the first question of survey of only designers

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 7 Skipped: 0



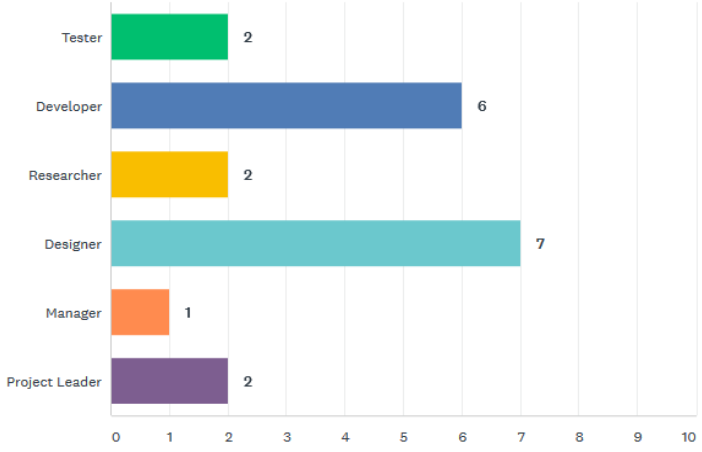
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	85.71%	6
I do code review but think comments aren't very helpful.	14.29%	1
I don't do any code reviews.	0.00%	0
TOTAL		7

Result of the second question of survey of only designers

Q3

Which areas you have experience on ?

Answered: 7 Skipped: 0



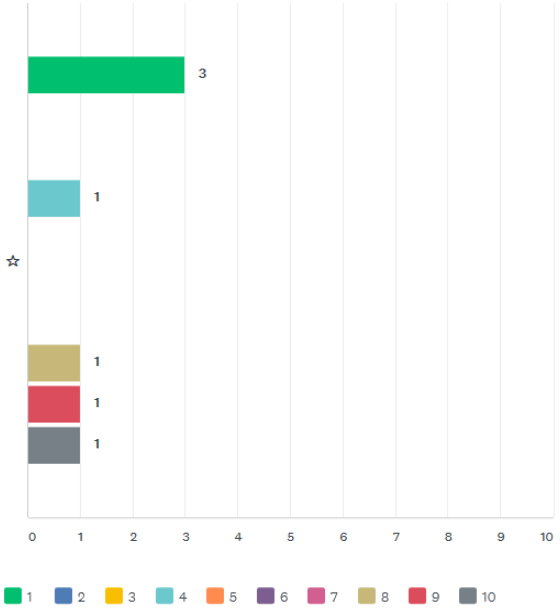
ANSWER CHOICES	RESPONSES	
Tester	28.57%	2
Developer	85.71%	6
Researcher	28.57%	2
Designer	100.00%	7
Manager	14.29%	1
Project Leader	28.57%	2
Total Respondents: 7		

Result of the third question of survey of only designers

Q4

Please rate this comment.

Answered: 7 Skipped: 0



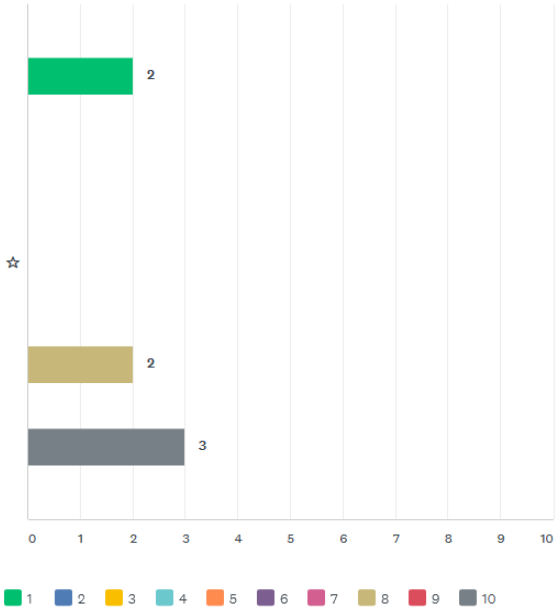
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	42.86% 3	0.00% 0	0.00% 0	14.29% 1	0.00% 0	0.00% 0	0.00% 0	14.29% 1	14.29% 1	14.29% 1	7	4.86

Result of the fourth question of survey of only designers

Q5

Please rate this comment.

Answered: 7 Skipped: 0



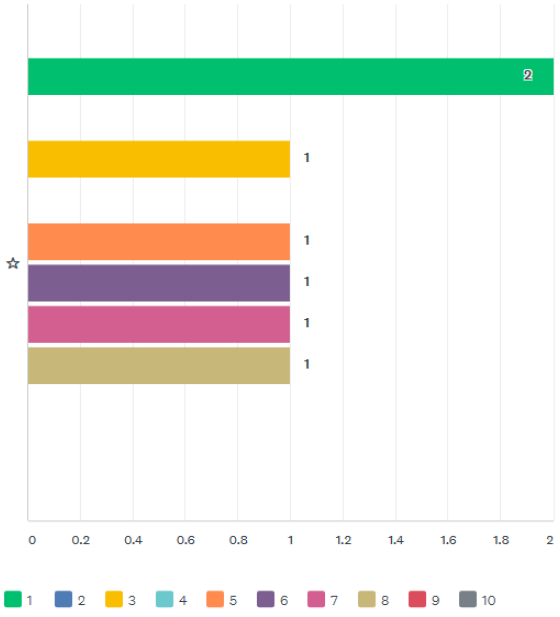
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	28.57% 2	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	28.57% 2	0.00% 0	42.86% 3	7	6.86

Result of the fifth question of survey of only designers

Q6

Please rate this comment.

Answered: 7 Skipped: 0



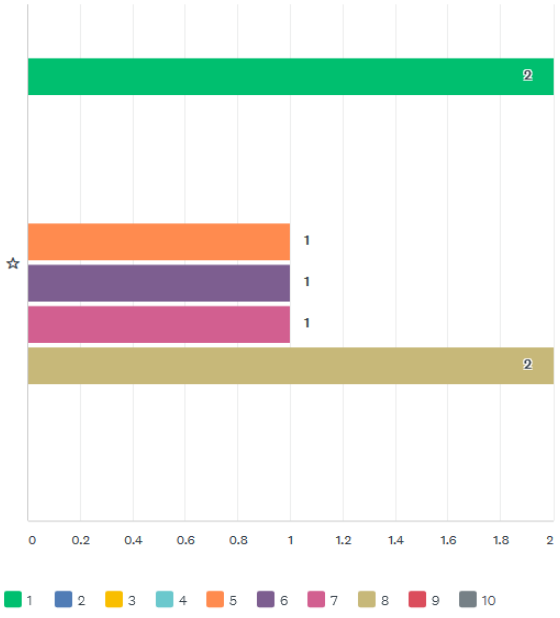
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	28.57% 2	0.00% 0	14.29% 1	0.00% 0	14.29% 1	14.29% 1	14.29% 1	14.29% 1	0.00% 0	0.00% 0	7	4.43

Result of the sixth question of survey of only designers

Q7

Please rate this comment.

Answered: 7 Skipped: 0



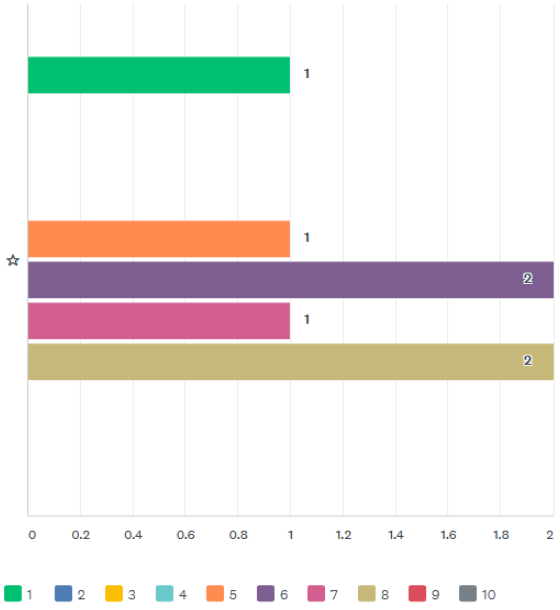
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	28.57% 2	0.00% 0	0.00% 0	0.00% 0	14.29% 1	14.29% 1	14.29% 1	28.57% 2	0.00% 0	0.00% 0	7	5.14

Result of the seventh question of survey of only designers

Q8

Please rate this comment.

Answered: 7 Skipped: 0



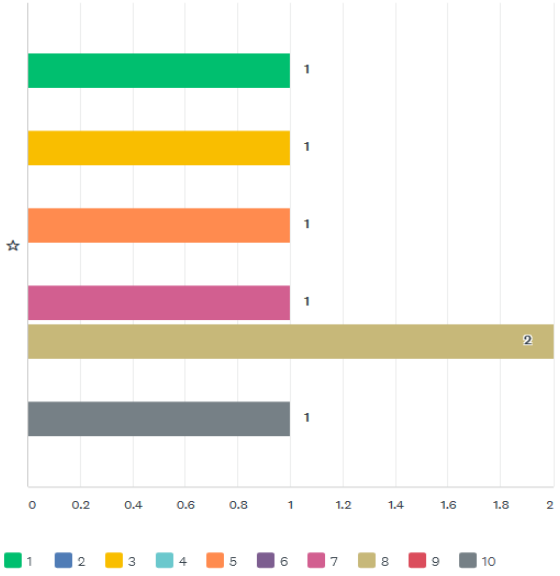
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	14.29% 1	0.00% 0	0.00% 0	0.00% 0	14.29% 1	28.57% 2	14.29% 1	28.57% 2	0.00% 0	0.00% 0	7	5.86

Result of the eighth question of survey of only designers

Q9

Please rate the overall quality of summary comments.

Answered: 7 Skipped: 0



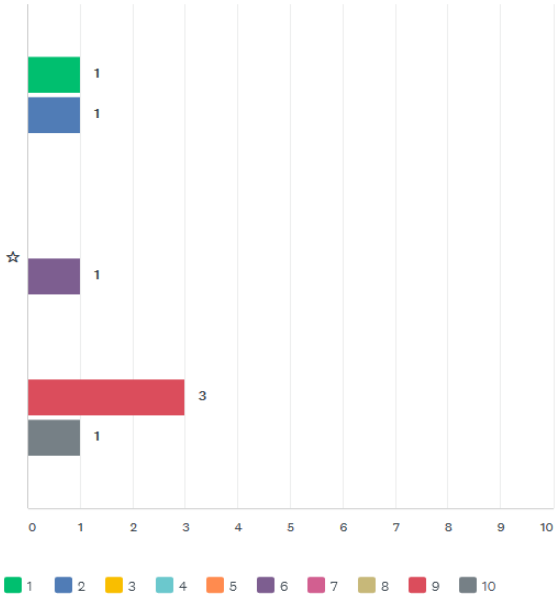
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	14.29% 1	0.00% 0	14.29% 1	0.00% 0	14.29% 1	0.00% 0	14.29% 1	28.57% 2	0.00% 0	14.29% 1	7	6.00

Result of the ninth question of survey of only designers

Q10

Please rate the overall quality of important statement comments.

Answered: 7 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	14.29% 1	14.29% 1	0.00% 0	0.00% 0	0.00% 0	14.29% 1	0.00% 0	0.00% 0	42.86% 3	14.29% 1	7	6.57

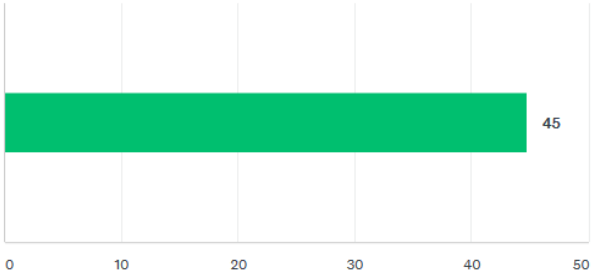
Result of the tenth question of survey of only designers

Survey Results of Developers

Q1

Work experience in months

Answered: 46 Skipped: 0



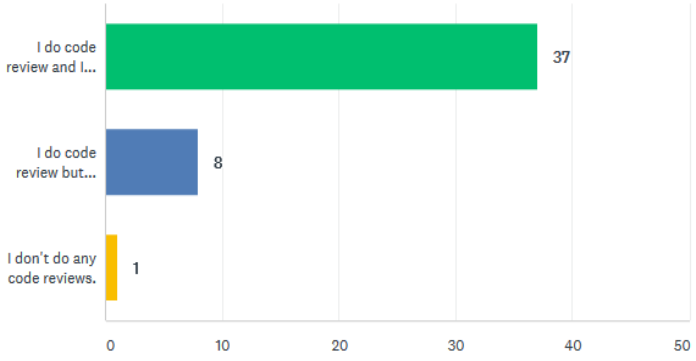
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	45	2,058	46
Total Respondents: 46			

Result of the first question of survey of only developers

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 46 Skipped: 0



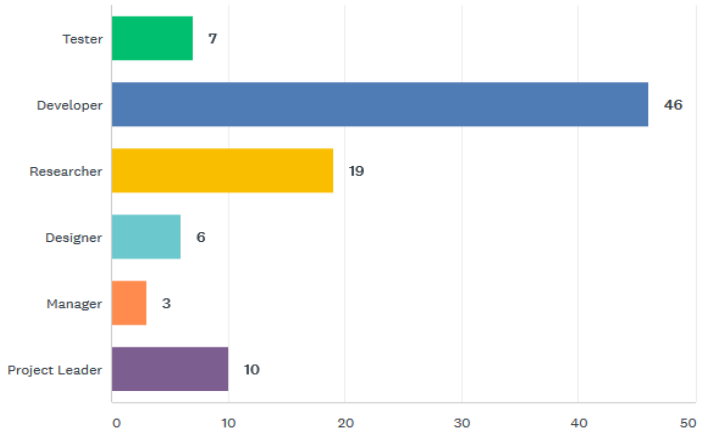
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	80.43%	37
I do code review but think comments aren't very helpful.	17.39%	8
I don't do any code reviews.	2.17%	1
TOTAL		46

Result of the second question of survey of only developers

Q3

Which areas you have experience on ?

Answered: 46 Skipped: 0



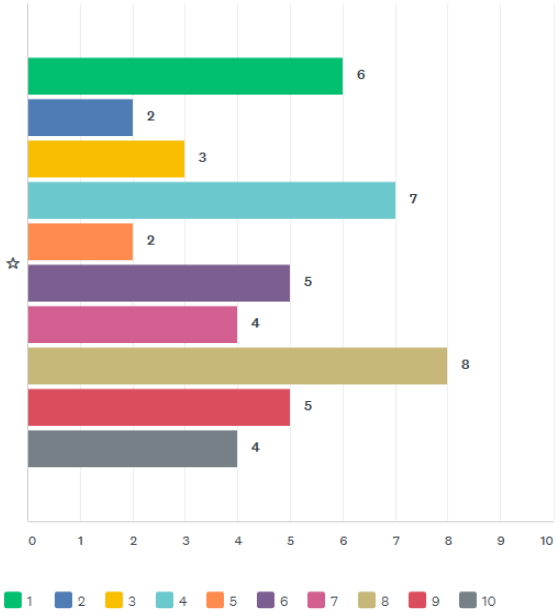
ANSWER CHOICES	RESPONSES	
Tester	15.22%	7
Developer	100.00%	46
Researcher	41.30%	19
Designer	13.04%	6
Manager	6.52%	3
Project Leader	21.74%	10
Total Respondents: 46		

Result of the third question of survey of only developers

Q4

Please rate this comment.

Answered: 46 Skipped: 0



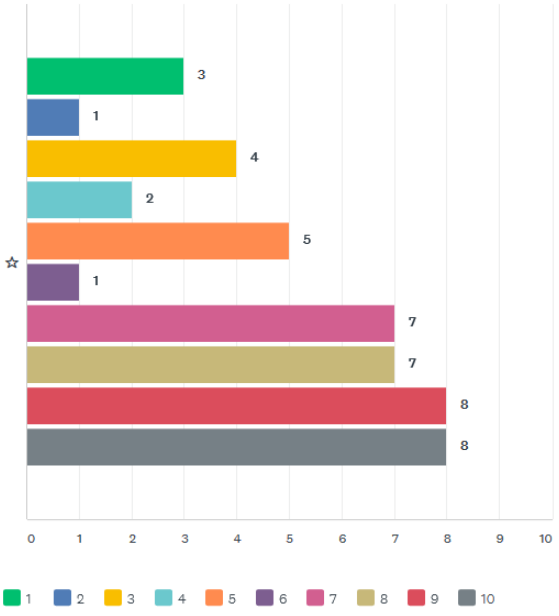
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	13.04% 6	4.35% 2	6.52% 3	15.22% 7	4.35% 2	10.87% 5	8.70% 4	17.39% 8	10.87% 5	8.70% 4	46	5.74

Result of the fourth question of survey of only developers

Q5

Please rate this comment.

Answered: 46 Skipped: 0



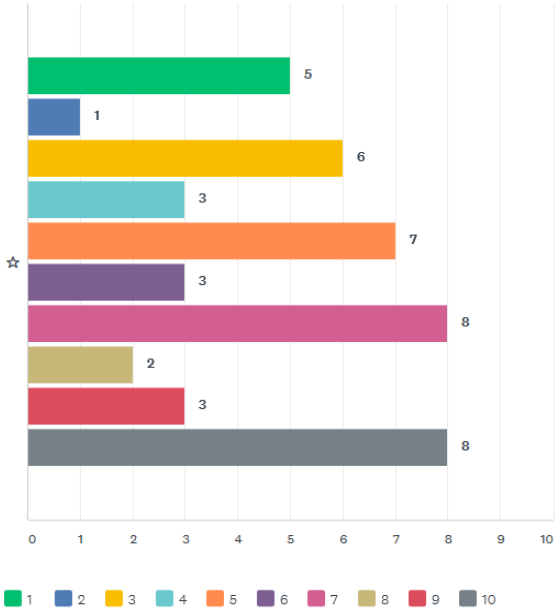
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	6.52% 3	2.17% 1	8.70% 4	4.35% 2	10.87% 5	2.17% 1	15.22% 7	15.22% 7	17.39% 8	17.39% 8	46	6.80

Result of the fifth question of survey of only developers

Q6

Please rate this comment.

Answered: 46 Skipped: 0



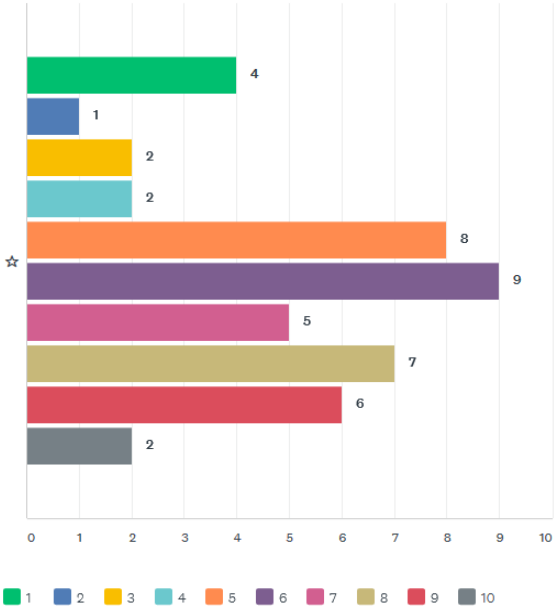
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	10.87% 5	2.17% 1	13.04% 6	6.52% 3	15.22% 7	6.52% 3	17.39% 8	4.35% 2	6.52% 3	17.39% 8	46	5.85

Result of the sixth question of survey of only developers

Q7

Please rate this comment.

Answered: 46 Skipped: 0



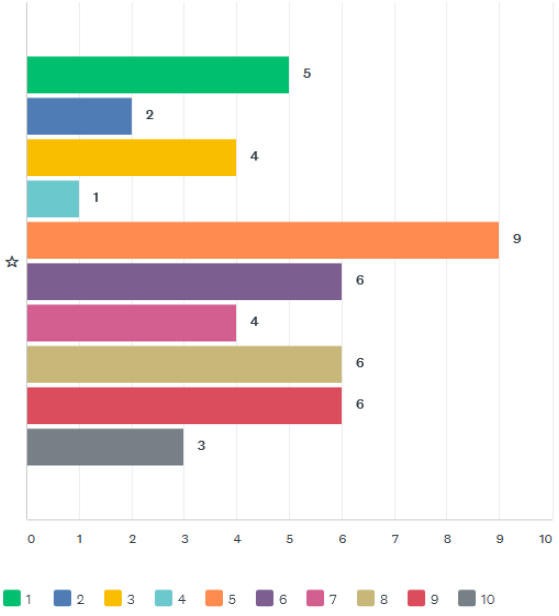
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	8.70% 4	2.17% 1	4.35% 2	4.35% 2	17.39% 8	19.57% 9	10.87% 5	15.22% 7	13.04% 6	4.35% 2	46	6.07

Result of the seventh question of survey of only developers

Q8

Please rate this comment.

Answered: 46 Skipped: 0



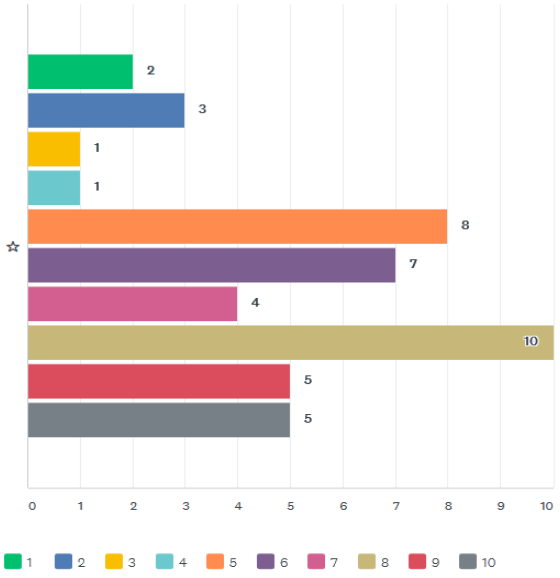
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	10.87% 5	4.35% 2	8.70% 4	2.17% 1	19.57% 9	13.04% 6	8.70% 4	13.04% 6	13.04% 6	6.52% 3	46	5.78

Result of the eighth question of survey of only developers

Q9

Please rate the overall quality of summary comments.

Answered: 46 Skipped: 0



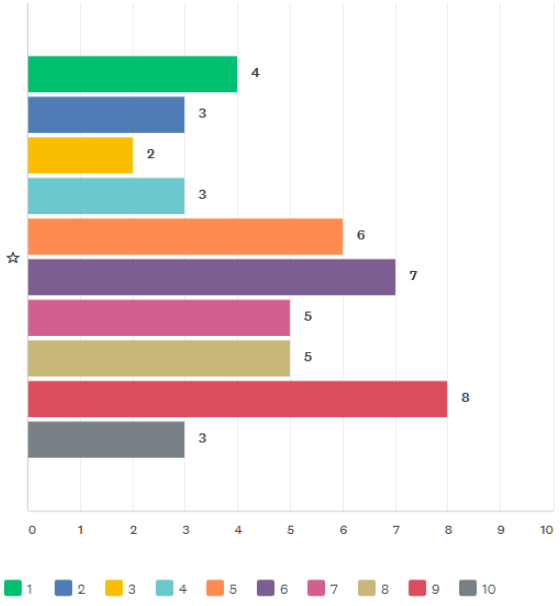
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.35% 2	6.52% 3	2.17% 1	2.17% 1	17.39% 8	15.22% 7	8.70% 4	21.74% 10	10.87% 5	10.87% 5	46	6.52

Result of the ninth question of survey of only developers

Q10

Please rate the overall quality of important statement comments.

Answered: 46 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	8.70% 4	6.52% 3	4.35% 2	6.52% 3	13.04% 6	15.22% 7	10.87% 5	10.87% 5	17.39% 8	6.52% 3	46	6.02

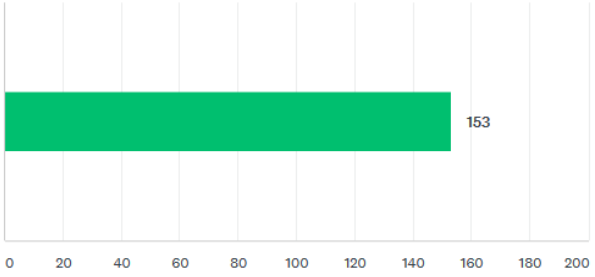
Result of the tenth question of survey of only developers

Survey Results of Managers

Q1

Work experience in months

Answered: 3 Skipped: 0



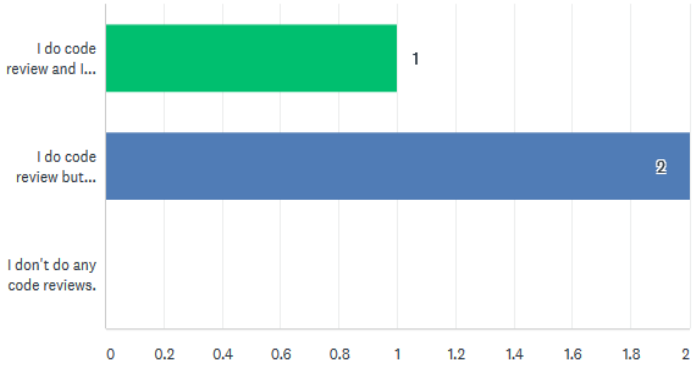
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	153	459	3
Total Respondents: 3			

Result of the first question of survey of only managers

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 3 Skipped: 0



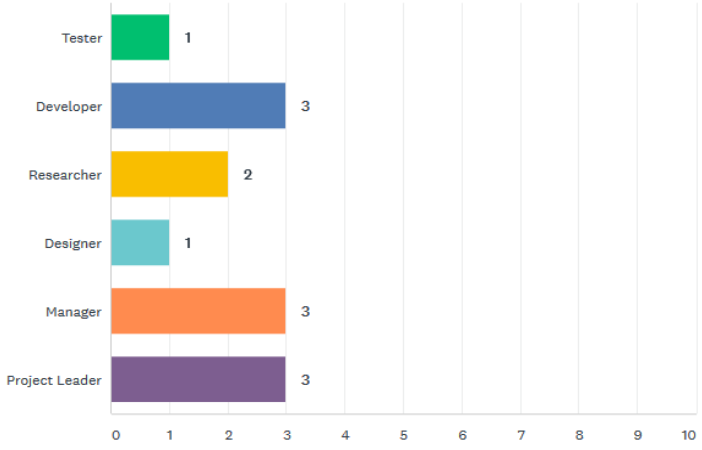
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	33.33%	1
I do code review but think comments aren't very helpful.	66.67%	2
I don't do any code reviews.	0.00%	0
TOTAL		3

Result of the second question of survey of only managers

Q3

Which areas you have experience on ?

Answered: 3 Skipped: 0



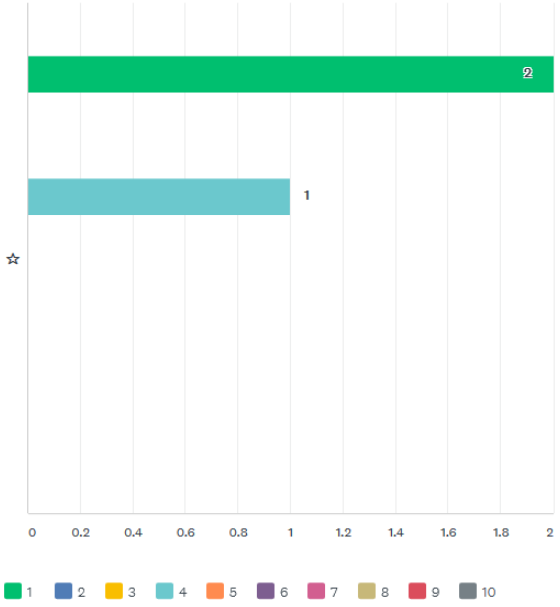
ANSWER CHOICES	RESPONSES	
Tester	33.33%	1
Developer	100.00%	3
Researcher	66.67%	2
Designer	33.33%	1
Manager	100.00%	3
Project Leader	100.00%	3
Total Respondents: 3		

Result of the third question of survey of only managers

Q4

Please rate this comment.

Answered: 3 Skipped: 0



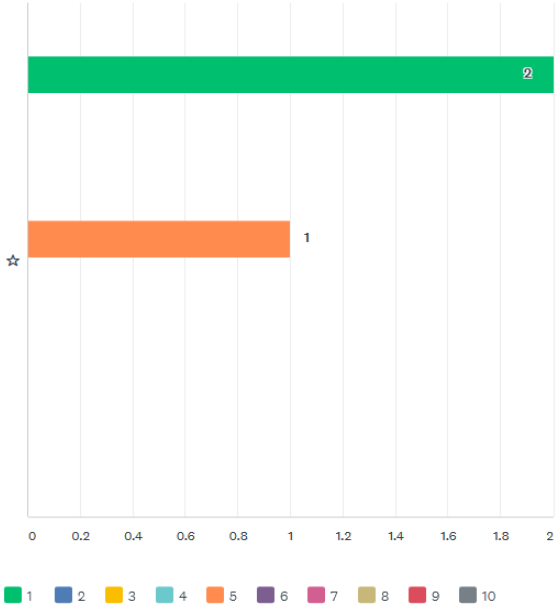
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.00

Result of the fourth question of survey of only managers

Q5

Please rate this comment.

Answered: 3 Skipped: 0



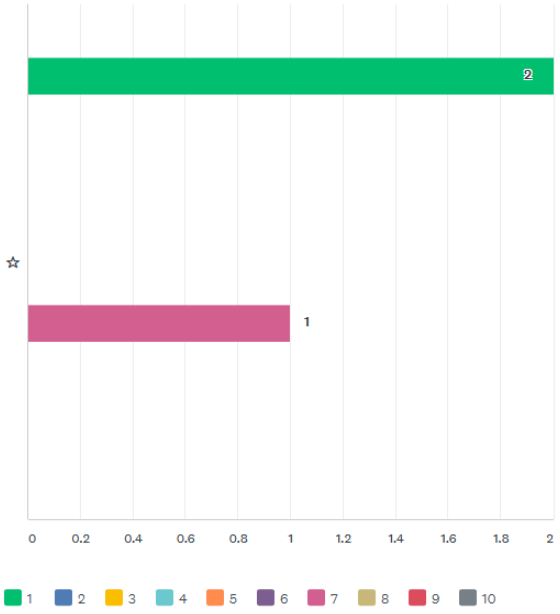
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.33

Result of the fifth question of survey of only managers

Q6

Please rate this comment.

Answered: 3 Skipped: 0



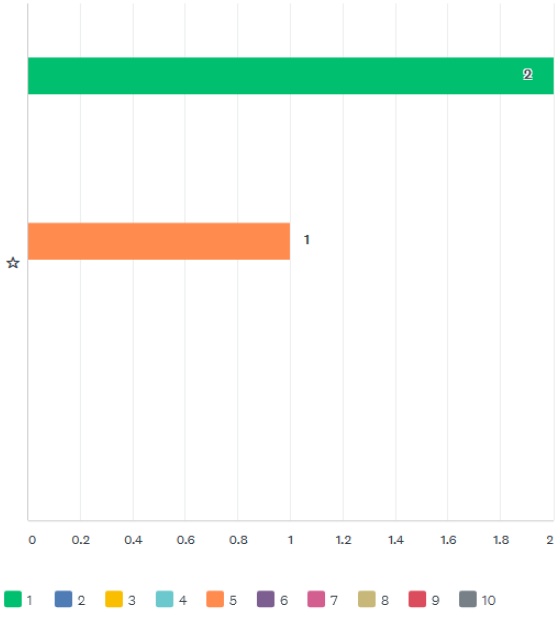
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	3	3.00

Result of the sixth question of survey of only managers

Q7

Please rate this comment.

Answered: 3 Skipped: 0



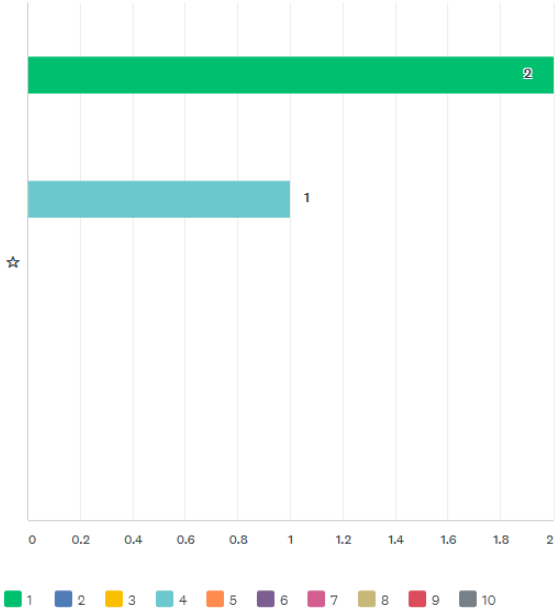
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.33

Result of the seventh question of survey of only managers

Q8

Please rate this comment.

Answered: 3 Skipped: 0



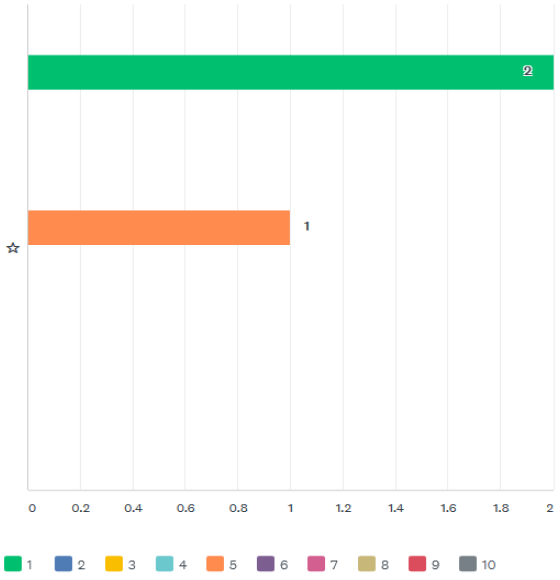
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.00

Result of the eighth question of survey of only managers

Q9

Please rate the overall quality of summary comments.

Answered: 3 Skipped: 0



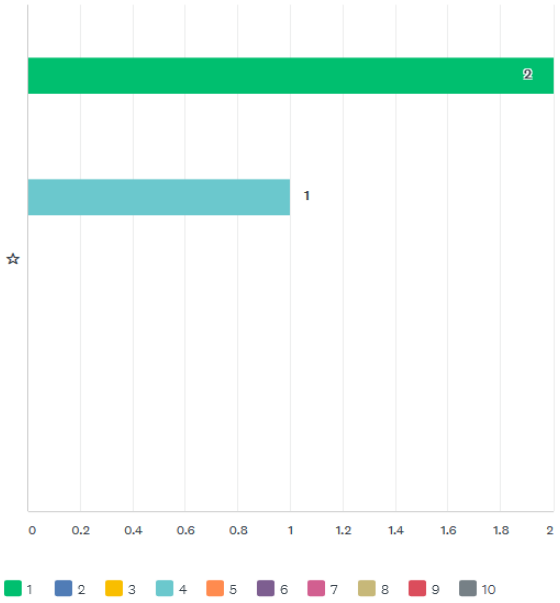
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.33

Result of the ninth question of survey of only managers

Q10

Please rate the overall quality of important statement comments.

Answered: 3 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	66.67% 2	0.00% 0	0.00% 0	33.33% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	0.00% 0	3	2.00

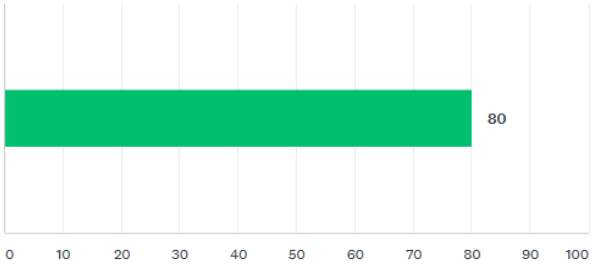
Result of the tenth question of survey of only managers

Survey Results of Project Leaders

Q1

Work experience in months

Answered: 12 Skipped: 0



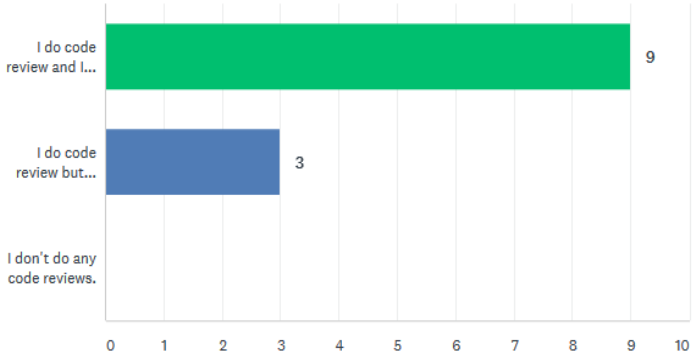
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	80	961	12
Total Respondents: 12			

Result of the first question of survey of only project leaders

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 12 Skipped: 0



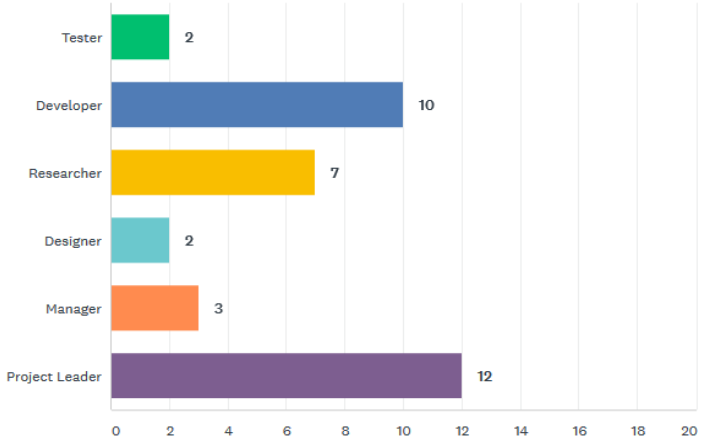
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	75.00%	9
I do code review but think comments aren't very helpful.	25.00%	3
I don't do any code reviews.	0.00%	0
TOTAL		12

Result of the second question of survey of only project leaders

Q3

Which areas you have experience on ?

Answered: 12 Skipped: 0



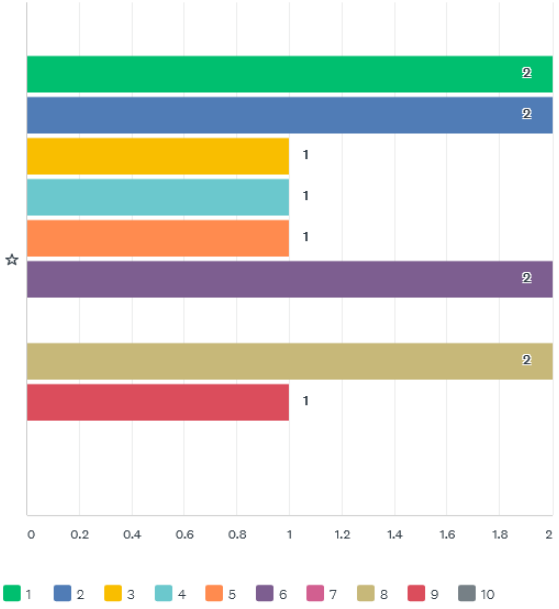
ANSWER CHOICES	RESPONSES	
Tester	16.67%	2
Developer	83.33%	10
Researcher	58.33%	7
Designer	16.67%	2
Manager	25.00%	3
Project Leader	100.00%	12
Total Respondents: 12		

Result of the third question of survey of only project leaders

Q4

Please rate this comment.

Answered: 12 Skipped: 0



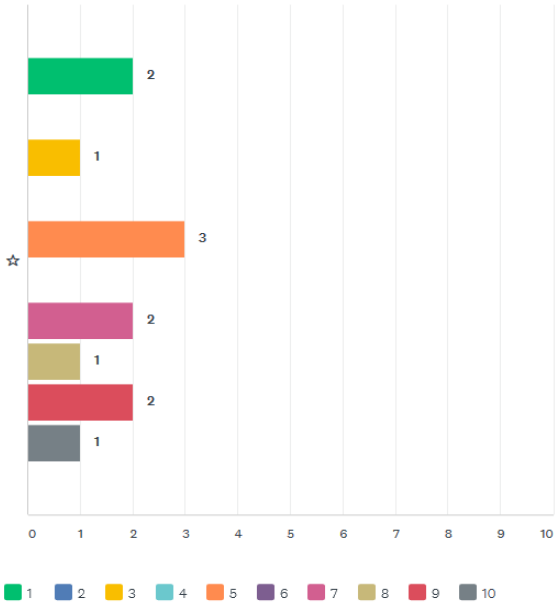
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	16.67% 2	8.33% 1	8.33% 1	8.33% 1	16.67% 2	0.00% 0	16.67% 2	8.33% 1	0.00% 0	12	4.58

Result of the fourth question of survey of only project leaders

Q5

Please rate this comment.

Answered: 12 Skipped: 0



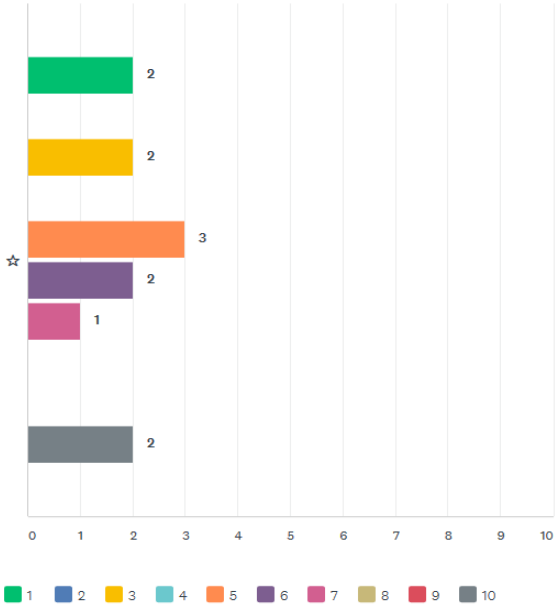
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	0.00% 0	8.33% 1	0.00% 0	25.00% 3	0.00% 0	16.67% 2	8.33% 1	16.67% 2	8.33% 1	12	5.83

Result of the fifth question of survey of only project leaders

Q6

Please rate this comment.

Answered: 12 Skipped: 0



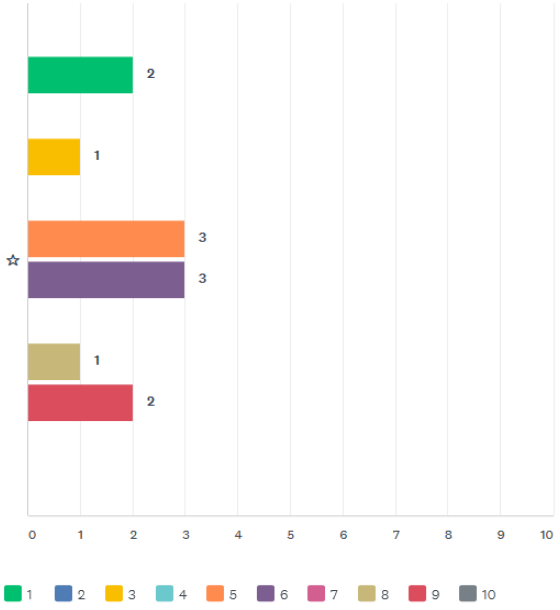
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	0.00% 0	16.67% 2	0.00% 0	25.00% 3	16.67% 2	8.33% 1	0.00% 0	0.00% 0	16.67% 2	12	5.17

Result of the sixth question of survey of only project leaders

Q7

Please rate this comment.

Answered: 12 Skipped: 0



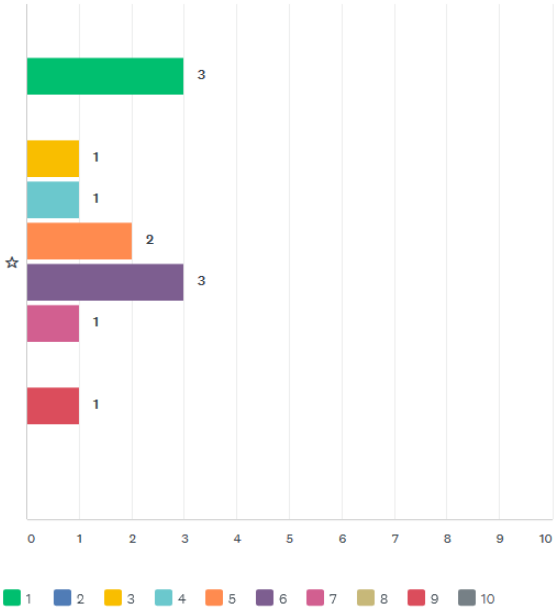
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	0.00% 0	8.33% 1	0.00% 0	25.00% 3	25.00% 3	0.00% 0	8.33% 1	16.67% 2	0.00% 0	12	5.33

Result of the seventh question of survey of only project leaders

Q8

Please rate this comment.

Answered: 12 Skipped: 0



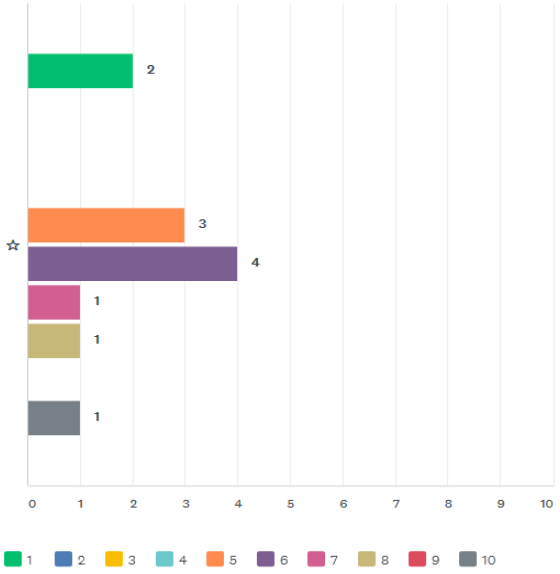
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	25.00% 3	0.00% 0	8.33% 1	8.33% 1	16.67% 2	25.00% 3	8.33% 1	0.00% 0	8.33% 1	0.00% 0	12	4.50

Result of the eighth question of survey of only project leaders

Q9

Please rate the overall quality of summary comments.

Answered: 12 Skipped: 0



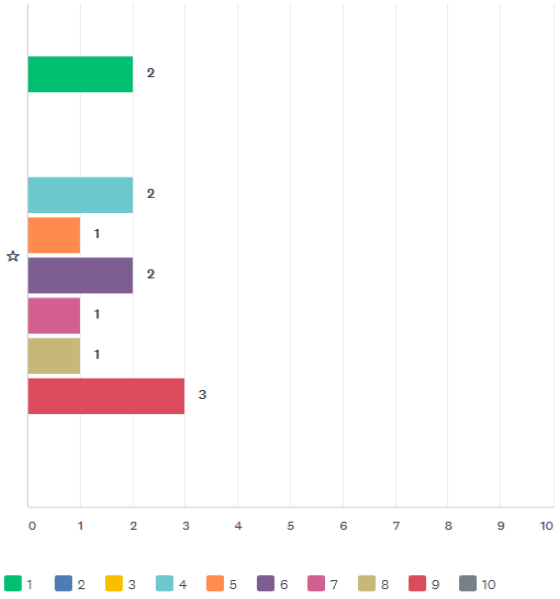
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	0.00% 0	0.00% 0	0.00% 0	25.00% 3	33.33% 4	8.33% 1	8.33% 1	0.00% 0	8.33% 1	12	5.50

Result of the ninth question of survey of only project leaders

Q10

Please rate the overall quality of important statement comments.

Answered: 12 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	16.67% 2	0.00% 0	0.00% 0	16.67% 2	8.33% 1	16.67% 2	8.33% 1	8.33% 1	25.00% 3	0.00% 0	12	5.75

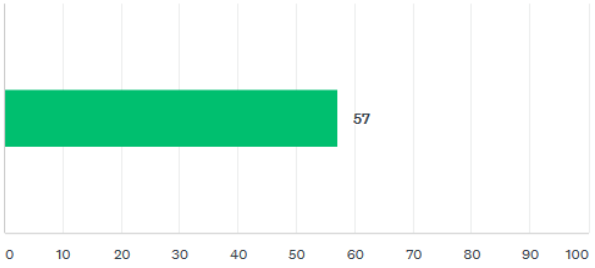
Result of the tenth question of survey of only project leaders

Survey Results of Researchers

Q1

Work experience in months

Answered: 21 Skipped: 0



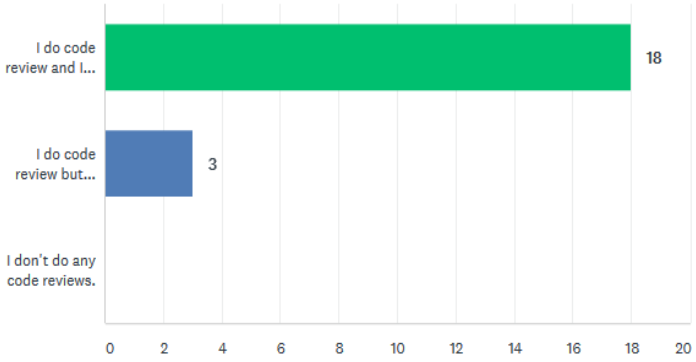
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	57	1,201	21
Total Respondents: 21			

Result of the first question of survey of only researchers

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 21 Skipped: 0



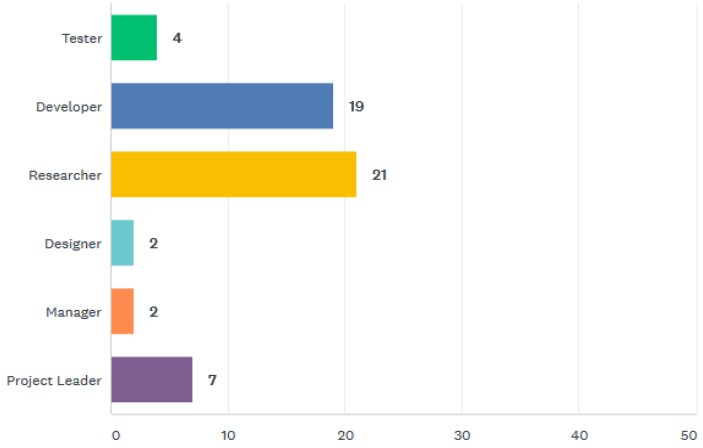
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	85.71%	18
I do code review but think comments aren't very helpful.	14.29%	3
I don't do any code reviews.	0.00%	0
TOTAL		21

Result of the second question of survey of only researchers

Q3

Which areas you have experience on ?

Answered: 21 Skipped: 0



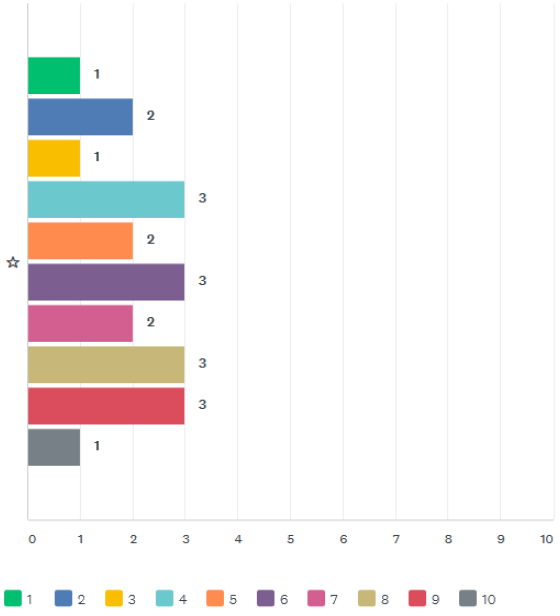
ANSWER CHOICES	RESPONSES	
Tester	19.05%	4
Developer	90.48%	19
Researcher	100.00%	21
Designer	9.52%	2
Manager	9.52%	2
Project Leader	33.33%	7
Total Respondents: 21		

Result of the third question of survey of only researchers

Q4

Please rate this comment.

Answered: 21 Skipped: 0



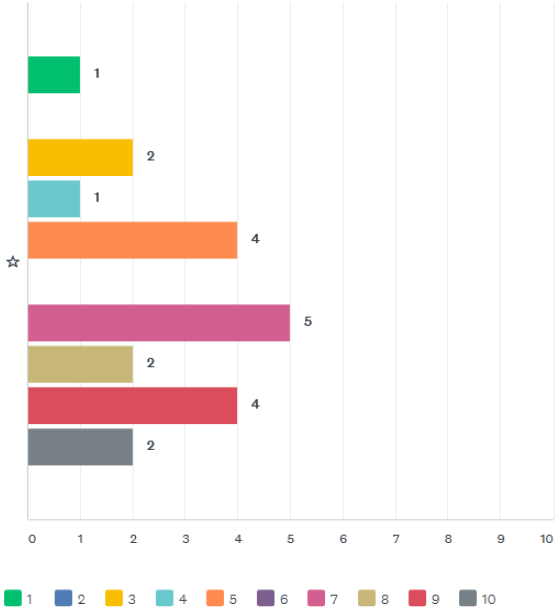
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	9.52% 2	4.76% 1	14.29% 3	9.52% 2	14.29% 3	9.52% 2	14.29% 3	14.29% 3	4.76% 1	21	5.86

Result of the fourth question of survey of only researchers

Q5

Please rate this comment.

Answered: 21 Skipped: 0



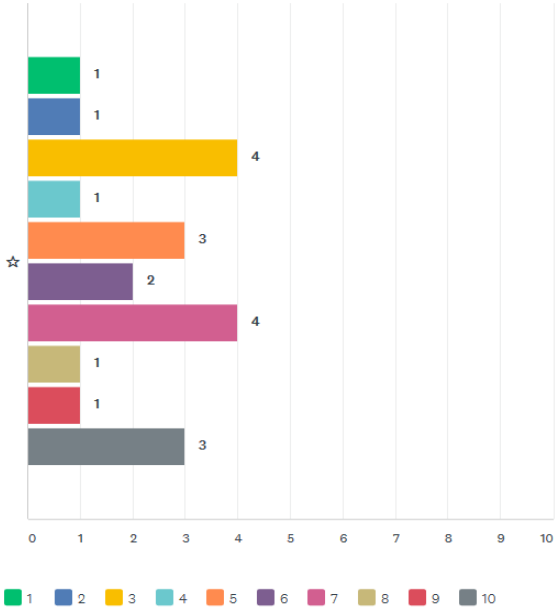
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	0.00% 0	9.52% 2	4.76% 1	19.05% 4	0.00% 0	23.81% 5	9.52% 2	19.05% 4	9.52% 2	21	6.57

Result of the fifth question of survey of only researchers

Q6

Please rate this comment.

Answered: 21 Skipped: 0



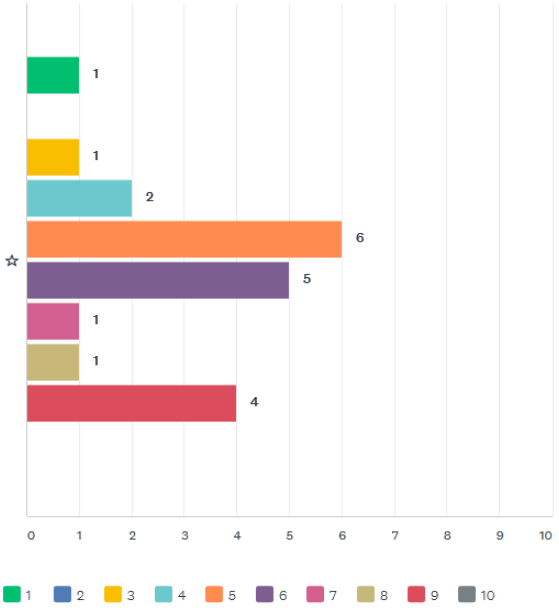
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	4.76% 1	19.05% 4	4.76% 1	14.29% 3	9.52% 2	19.05% 4	4.76% 1	4.76% 1	14.29% 3	21	5.76

Result of the sixth question of survey of only researchers

Q7

Please rate this comment.

Answered: 21 Skipped: 0



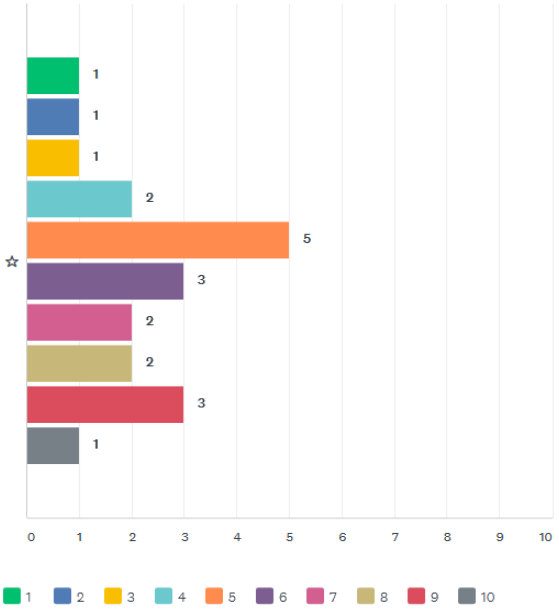
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	0.00% 0	4.76% 1	9.52% 2	28.57% 6	23.81% 5	4.76% 1	4.76% 1	19.05% 4	0.00% 0	21	5.86

Result of the seventh question of survey of only researchers

Q8

Please rate this comment.

Answered: 21 Skipped: 0



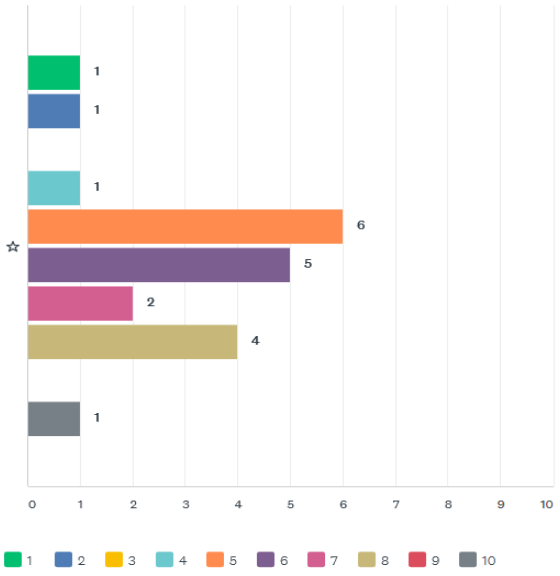
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	4.76% 1	4.76% 1	9.52% 2	23.81% 5	14.29% 3	9.52% 2	9.52% 2	14.29% 3	4.76% 1	21	5.90

Result of the eighth question of survey of only researchers

Q9

Please rate the overall quality of summary comments.

Answered: 21 Skipped: 0



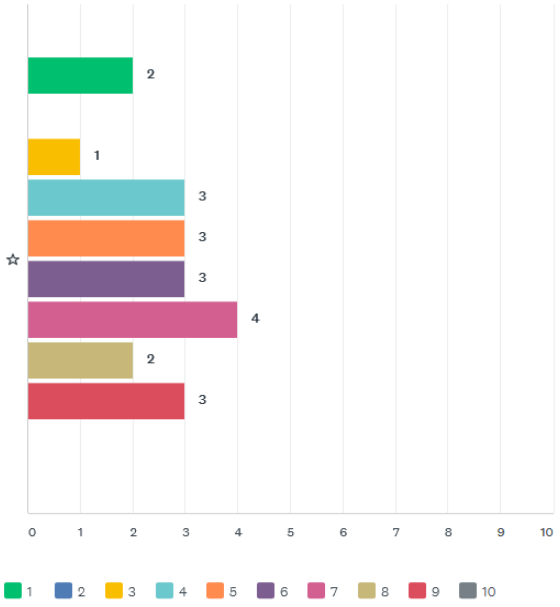
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	4.76% 1	4.76% 1	0.00% 0	4.76% 1	28.57% 6	23.81% 5	9.52% 2	19.05% 4	0.00% 0	4.76% 1	21	5.86

Result of the ninth question of survey of only researchers

Q10

Please rate the overall quality of important statement comments.

Answered: 21 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	9.52% 2	0.00% 0	4.76% 1	14.29% 3	14.29% 3	14.29% 3	19.05% 4	9.52% 2	14.29% 3	0.00% 0	21	5.76

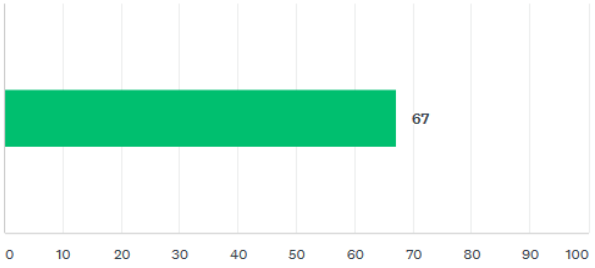
Result of the tenth question of survey of only researchers

Survey Results of Testers

Q1

Work experience in months

Answered: 9 Skipped: 0



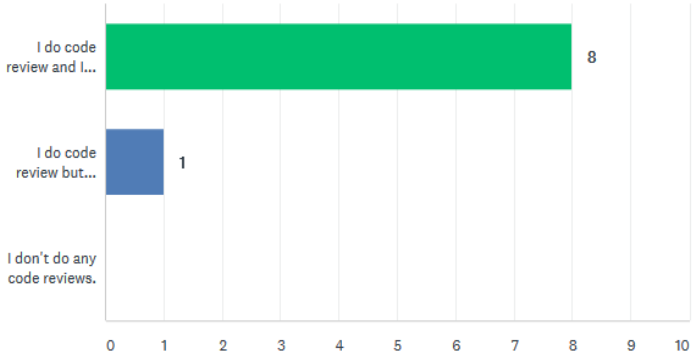
ANSWER CHOICES	AVERAGE NUMBER	TOTAL NUMBER	RESPONSES
Responses	67	604	9
Total Respondents: 9			

Result of the first question of survey of only testers

Q2

Do you do code review? If so, then do you think that comments are helpful for code review?

Answered: 9 Skipped: 0



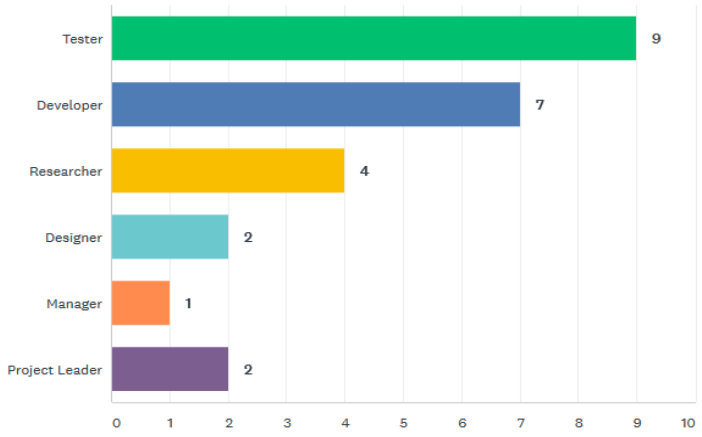
ANSWER CHOICES	RESPONSES	
I do code review and I think comments are helpful	88.89%	8
I do code review but think comments aren't very helpful.	11.11%	1
I don't do any code reviews.	0.00%	0
TOTAL		9

Result of the second question of survey of only testers

Q3

Which areas you have experience on ?

Answered: 9 Skipped: 0



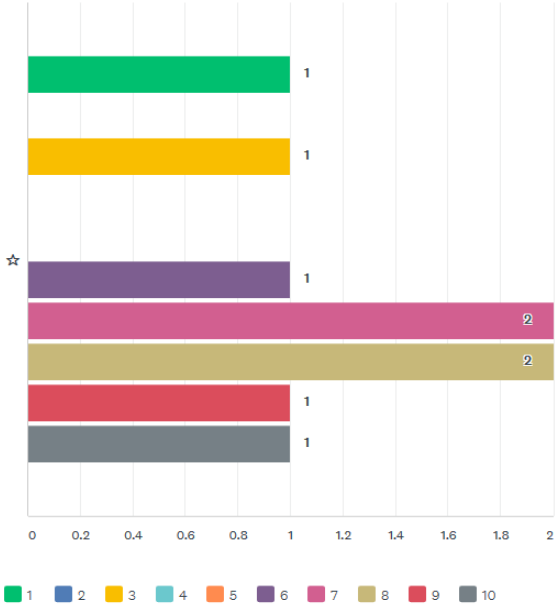
ANSWER CHOICES	RESPONSES	
Tester	100.00%	9
Developer	77.78%	7
Researcher	44.44%	4
Designer	22.22%	2
Manager	11.11%	1
Project Leader	22.22%	2
Total Respondents: 9		

Result of the third question of survey of only testers

Q4

Please rate this comment.

Answered: 9 Skipped: 0



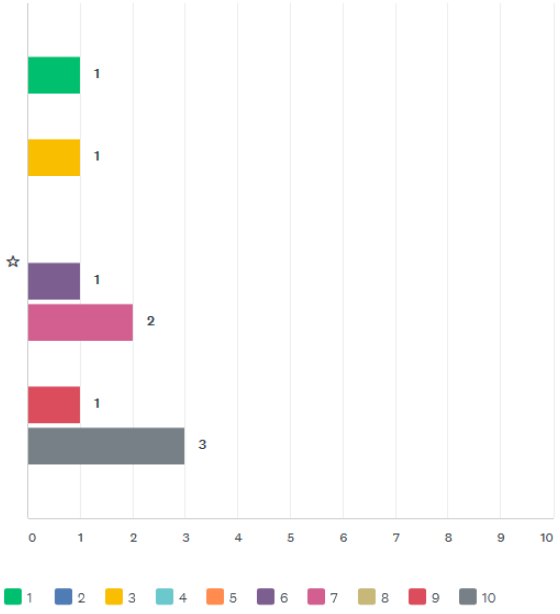
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	11.11% 1	0.00% 0	0.00% 0	11.11% 1	22.22% 2	22.22% 2	11.11% 1	11.11% 1	9	6.56

Result of the fourth question of survey of only testers

Q5

Please rate this comment.

Answered: 9 Skipped: 0



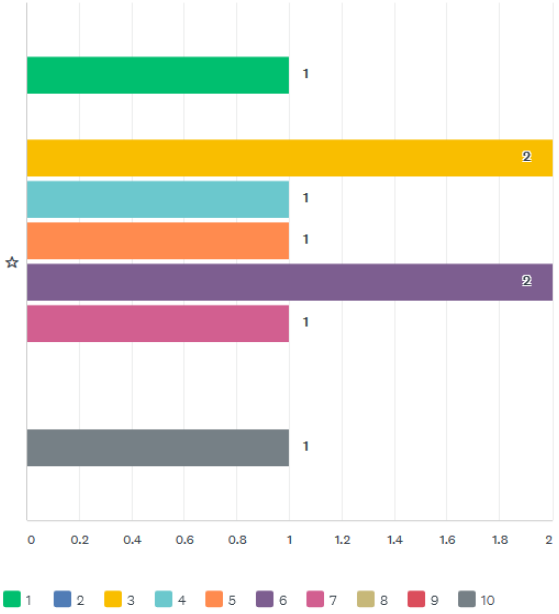
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	11.11% 1	0.00% 0	0.00% 0	11.11% 1	22.22% 2	0.00% 0	11.11% 1	33.33% 3	9	7.00

Result of the fifth question of survey of only testers

Q6

Please rate this comment.

Answered: 9 Skipped: 0



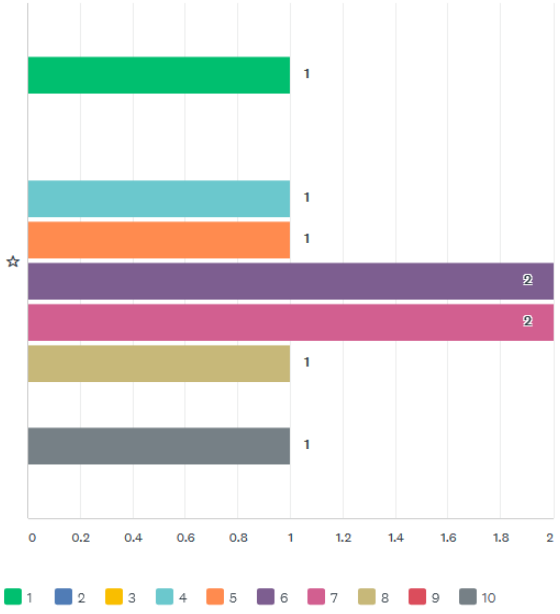
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	22.22% 2	11.11% 1	11.11% 1	22.22% 2	11.11% 1	0.00% 0	0.00% 0	11.11% 1	9	5.00

Result of the sixth question of survey of only testers

Q7

Please rate this comment.

Answered: 9 Skipped: 0



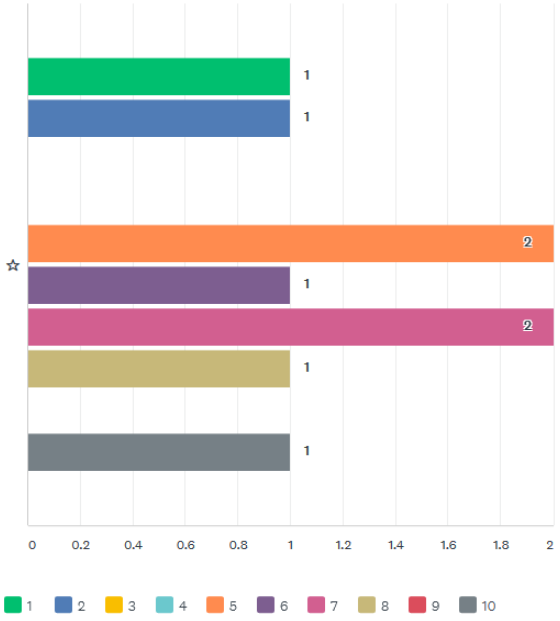
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	0.00% 0	11.11% 1	11.11% 1	22.22% 2	22.22% 2	11.11% 1	0.00% 0	11.11% 1	9	6.00

Result of the seventh question of survey of only testers

Q8

Please rate this comment.

Answered: 9 Skipped: 0



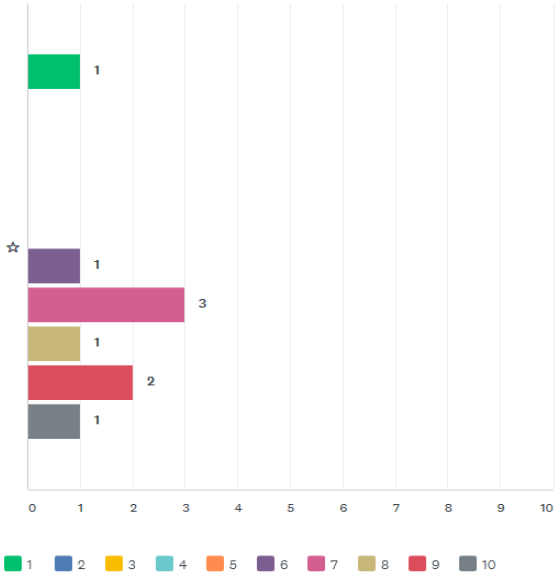
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	11.11% 1	0.00% 0	0.00% 0	22.22% 2	11.11% 1	22.22% 2	11.11% 1	0.00% 0	11.11% 1	9	5.67

Result of the eighth question of survey of only testers

Q9

Please rate the overall quality of summary comments.

Answered: 9 Skipped: 0



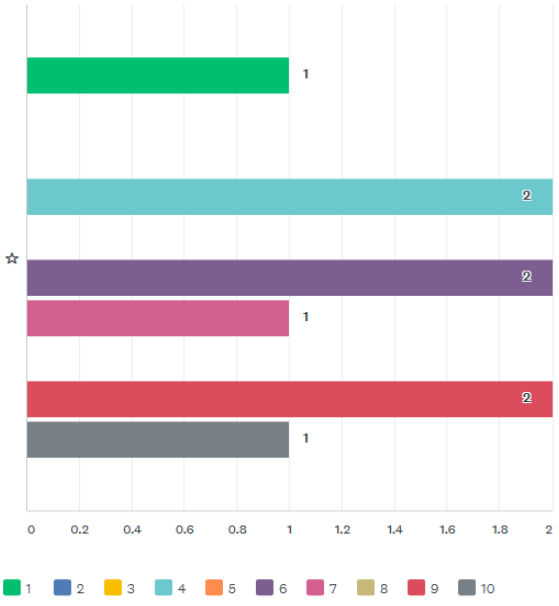
	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	0.00% 0	0.00% 0	0.00% 0	11.11% 1	33.33% 3	11.11% 1	22.22% 2	11.11% 1	9	7.11

Result of the ninth question of survey of only testers

Q10

Please rate the overall quality of important statement comments.

Answered: 9 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	TOTAL	WEIGHTED AVERAGE
☆	11.11% 1	0.00% 0	0.00% 0	22.22% 2	0.00% 0	22.22% 2	11.11% 1	0.00% 0	22.22% 2	11.11% 1	9	6.22

Result of the tenth question of survey of only testers

References

[1] "Chapter 2. grammars," 2017. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html

[2] "Chapter 18. syntax," 2017. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html

[3] "Code conventions for the java programming language," 2017. [Online]. Available: http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

[4] L. Zhang, T. Qin, Z. Zhou, D. Hao, and J. Sun, "Identifying use cases in source code," *Journal of Systems and Software*, vol. 79, no. 11, pp. 1588–1598, 2006.

[5] B. Fluri, M. Würsch, E. Giger, and H. C. Gall, "Analyzing the co-evolution of comments and source code," *Software Quality Journal*, vol. 17, no. 4, pp. 367–394, 2009.

[6] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/1858996.1859006

[7] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Automatically detecting and describing high level actions within methods," *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, 2011.

[8] ——, "Generating parameter comments and integrating with method summaries," *2011 IEEE 19th International Conference on Program Comprehension*, 2011.

[9] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," *2010 17th Working Conference on Reverse Engineering*, 2010.

[10] L. Moreno, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Jsummarizer: An automatic generator of natural language summaries for java classes," *2013 21st International Conference on Program Comprehension (ICPC)*, 2013.

[11] L. Moreno, "Summarization of complex software artifacts," *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 2014.

[12] P. W. Mcburney and C. Mcmillan, "Automatic documentation generation via source code summarization of method context," *Proceedings of the 22nd International Conference on Program Comprehension - ICPC 2014*, 2014.

[13] E. Yildiz and E. Ekin, "Automatic comment generation using only source code," in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, May 2017, pp. 1–4.

[14] R. Pawlak, M. Monperrus, N. Petitprez, C. Noguera, and L. Seinturier, "Spoon: A library for implementing analyses and transformations of java source code," *Software: Practice and Experience*, vol. 46, pp. 1155–1179, 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01078532/document

[15] "Defining methods, the java tutorials learning the java language classes and objects," 2017. [Online]. Available: https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

[16] "Introduction to collections the java tutorials collections," 2017. [Online]. Available: https://docs.oracle.com/javase/tutorial/collections/intro/index.html

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[18] 2017. [Online]. Available: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

[19] T. Parr, *The Definitive ANTLR 4 Reference*, 2nd ed. Pragmatic Bookshelf, 2013.

[20] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[21] "Android documentation of activity class," 2017. [Online]. Available: https://developer.android.com/reference/android/app/Activity.html

[22] "Surveymonkey," San Mateo, California, USA, 2017. [Online]. Available: https://www.surveymonkey.com