Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Literature review and concept for cooperation in software development

Literaturstudie und Konzept für Zusammenarbeit bei der Softwareentwicklung

Masterarbeit

im Studiengang Informatik

von

Meriem Haltiti

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Joel Greenyer
Betreuer: M. Sc. Melanie Busch

Hannover, 03. Juni 2019

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 03. Juni 2019

Meriem Haltiti

Zusammenfassung

Softwareprojekte werden immer komplexer. Mit steigende Anzahl der Methoden, Klassen und Objekten, steigt die Komplexität. Darüber hinaus tragen die verschiedenen Beziehungen zwischen den Elementen und die Anzahl der Mitarbeiter, die das gleiche Projekt bearbeiten und erweitern, der Komplexität bei. Demzufolge wird es schwierig, ein Softwareprojekt zu erfassen. Das Verständnis erfordert viel Expertise und Fachkenntnisse. Infolgedessen ist der Entwickler die meiste Zeit damit beschäftigt, einen unbekannten Code zu editieren und zu verstehen. Um den Überblick über das Projekt zu behalten, werden Ansätze verwendet, die den Entwicklern dabei unterstützen die Zusammenarbeit zu verbessern.

Diese Arbeit zielt darauf ab, alle diese Ansätze aus den letzten 15 Jahren der Literatur zusammenzufassen und zu bewerten. Diese Ansätze dienen der Lenkung der Aufmerksamkeit in der Zusammenarbeit des Entwicklers auf den wichtigen Teil des Codes oder des Dokuments. Danach werden diese Ansätze kategorisiert und untereinander verglichen. Außerdem werden die verschiedenen Techniken, mit denen die Aufmerksamkeit des Entwicklers gemessen wird, zusammengefasst. Die Wechselwirkung zwischen der Aufmerksamkeit des Entwicklers und der kollaborativen Programmierung wird somit erläutert. Die Visualisierung zeigt die Lücken und Gemeinsamkeiten in diesem Forschungsbereich auf. Der letzte Teil beinhaltet Diskussionen und Interpretationen, um Möglichkeiten für zukünftige Forschungen zu eröffnen.

Abstract

Software projects are becoming ever more complex. This complexity increases with the rising number of its different methods, classes, and objects. Besides, the different relationships between each component and the number of the employees add to the complexity. Each developer is expanding, changing, fixing, and reviewing the same project. As a consequence, the software project becomes difficult to grasp. A lot of knowledge and expertise are needed to understand it. Therefore, the developer spends most of his time manipulating and comprehending an unfamiliar code. To keep track of the project, approaches are used by the developer to enhance the collaboration between the team members.

This thesis aims to review and synthesize all these approaches that guide the attention of the developer toward the relevant part in the code or the document at the cooperation spanning the past 15 years of literature. After that, these approaches are categorized, then compared. Additionally, the different techniques for measuring the attention of the developer are summarized, and the reciprocity between the attention of the developer and collaborative programming is explained. Visualizations are designed to highlight the gaps and similarities in this research area. In the end, a discussion and an interpretation are held in order to open opportunities for future research.

Acronyms

AOI	Area Of Interest
CAISE	Collaborative Architecture for Iterative Software Engineering
CRI	Continuum of Relevance Index
CSE	Collaborative Software Engineering
CVS	Concurrent Versioning System
EBSE	Evidence-Based Software Engineering
EEG	Electroencephalographie
FASTDash	Fostering Awarness for Software Teams Dashboard
fMRI	functional Magnetic Resonance Imaging
IDE	Integrated Development Environment
PROM	PRO Metrics
SLR	Systematic Literature Review
TeamWATCH	Team-based Workspace Awarness Toolkit and Collaboration Hub
TTFF	Time To First Fixation
UML	Unified Modeling Language

Contents

Chapter 1

Introduction

1.1 Motivation

The environment of software engineering is continuously changing. From simple keystrokes to sophisticated functional applications coupling many interlocking pieces such as objects, classes and methods, software systems are developed. The target of software engineering is to decompose sophisticated functionality into independent although connected modules. The modules are used in order to speed up the process of the development. They are done to avoid reimplementing the same code for similar functions in every new project. Henceforth, this procedure involves collaboration [35]. Software development is not only collaborative but also a knowledge-intensive process [41]. Besides, the long duration of the project, different employees such as developer, software architect, stakeholder, etc., make the knowledge transfer problematic.

Moreover, it is challenging to verbally transfer knowledge and expertise. However, a major problem with this kind of process is, spending the majority of the time, comprehending an unfamiliar code and seeking consistent information that may sometimes be finished unsuccessfully. The lack of tracking of all the relevant processes of the development, makes the software development task difficult. The programmer cannot follow all the changes and loses the overview of the accomplished and not accomplished tasks. Also in most cases, he is confronted with editing or debugging an unfamiliar code. This challenging environment pushes the developer to regularly rethink and adapt their development processes to new conditions in order to stay aware

of all changes and recommendations.

As this continuous change becomes the norm, the software developer has to use approaches that support him to focus on the relevant part in the document or the code. Furthermore, this support helps him to maintain an overview of the complete project, to enhance their program understanding and to assure him more flexibility in the team.

1.2 Goal

The main of this thesis is to provide a literature review that documents several key contributions made to the fields of software engineering in steering the attention of the developer during software development. The goal of this contributions is to enhance the program comprehension and maintain an overview of the important activities during collaborative programming. Secondly, gaps and similarities between these retrieved approaches will be identified visualized, and then assessed.

1.3 Scope of the research

This thesis focuses on determining all of the approaches that are used to steer the attention of the developer during collaborative programming. After that, these approaches are classified through the art of sharing the attention of the developer. Following this, an evaluation of these approaches is done and a visualization is conceived to give an outline of the similarities and gaps of the retrieved approaches. Last but not least, conclusions and implications are also presented to support several possible future directions in this field.

1.4 Stucture of the research

The overall structure of the study takes the form of seven chapters, including this introductory chapter. Chapter two begins by laying out the theoretical dimensions of the research, and looks at how key terms are defined in the literature review. The third chapter is concerned with the methodology used for this study, which is systematic literature review with snowballing procedure. The fourth chapter describes the followed review methods and the analysis of the retrieved papers. The fifth chapter presents the findings of

the research, focusing on the three key themes that are: classification of the approaches, attention measurement during collaborative programming and the correlation between collaboration and the attention of the developer. The sixth chapter is designed for the implications and discussion. The final chapter draws upon the entire thesis, tying up the followed steps in this review and the important findings as well as an outlook to future research into this area.

Chapter 2

Theory

The aim of this chapter is to shed light on the theoretical approaches that found this thesis. The basis of systematic literature review and snowballing research procedures will be described. Next, collaboration in software engineering and program comprehension will be explained.

2.1 Systematic literature review

A Systematic Literature Review (SLR) is a form of secondary study, which refers to identifying, evaluating and interpreting of all primary studies that are related to a particular research question, topic area or a phenomenon. Primary studies refer to the studies that contribute to the systematic review [26].

It uses the evidence-based paradigm in software engineering. Kitchenham proposes the introduction of this paradigm. The aim of Evidence-Based Software Engineering (EBSE) is recapitulated by Kitchenham [19], as follows: "to provide how current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software".

Therefore, this review makes a significant contribution to provide reliable and unbiased results. The main advantage of SLR is not only summarising all available approaches that are related to a specific field, but also identifying all gaps and similarities of these approaches. The SLR is a background for further investigations that can be used by other researchers to gain time and effort [26]. SLR is distinguished over the conventional form of

literature review through different features such as the definition of the research questions, inclusion, and exclusion criteria in the beginning of the review, the specification of the search strategy of the primary studies, the documentation of all the steps and the assessment of all primary studies.

According to Kitchenham [26], the steps of SLR are divided into three major stages. They are illustrated in figure 2.1.

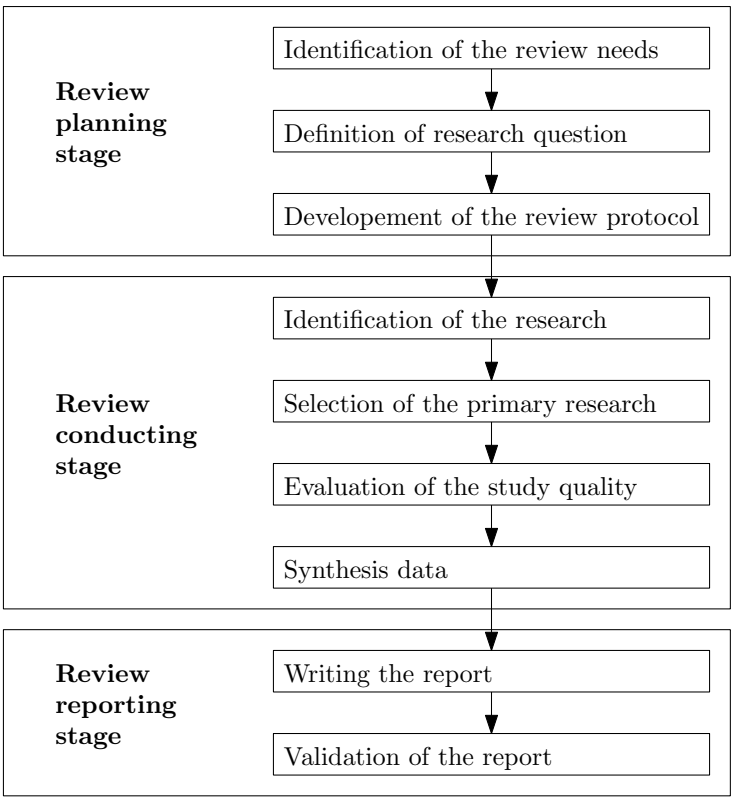

Figure 2.1: Systematic Literature Review process based on Kitchenham guidelines [26]

Each stage of the figure 2.1 will be discussed in detail in the following step [26].

Stage 1: Review planning

The first stage is based on understanding the need for the review. Research questions and validation of the review protocol are done in this step. The keywords will be extracted from the research questions. This

phase is crucial because it helps the reviewer to stay in the subject. The protocol will include the background, context of the research, search strategy and selection criteria of the selected papers [26].

Stage 2: Review conducting

The second stage is based on the search of the evidence to answer the research questions, and then to identify the primary studies. In the following step, the study quality is assessed.

The search consists of inserting the search string with the predefined keywords in the selected database. The search string is combined from a Boolean operator such as "AND", "OR" and "NOT" and the keywords; synonyms of the keywords are also taken into account. After inserting the search-string in the selected databases, the retrieved papers will be assessed. Exclusion and inclusion criteria will be applied first through the title, second through the abstract, and then through the whole paper. This step is done to assure that only the relevant papers will be selected. The selected papers will be defined as the primary study and will be included in the review. Later, these primary studies will be summarized [26].

Stage 3: Review reporting

Last but not least, the third stage "Review reporting" consists of the outcome of all the previous steps which includes the writing and the validation of the report. The whole process can be repeated many times until a transparent, understandable, and reproducible review is reached. The iteration can be done primarily in the stage of planning the review so that inclusion and exclusion criteria can be redefined; the extraction of the required data can be done many times after applying the quality criteria [26].

2.2 Snowballing research procedure

The goodness of a SLR is dependent especially on the search procedure. The quality of the search is significant for the efficiency and the reliability of the retrieved papers [61].

In SLR, the search of all relevant papers is very challenging on account of the quality of the search string, the choice of the databases and island

papers. These criteria influence the results. That is why a search procedure should be used to support the exploration of all relevant papers. In this review, the snowballing search procedure is applied. It inheres to handling the reference list of a paper or citations of the paper to determine further papers. The search is finished when no new relevant papers are found. This search procedure is divided into two main phases: Start set, backward and forward snowballing [61]. The illustration of this procedure can be seen in figure 2.2.

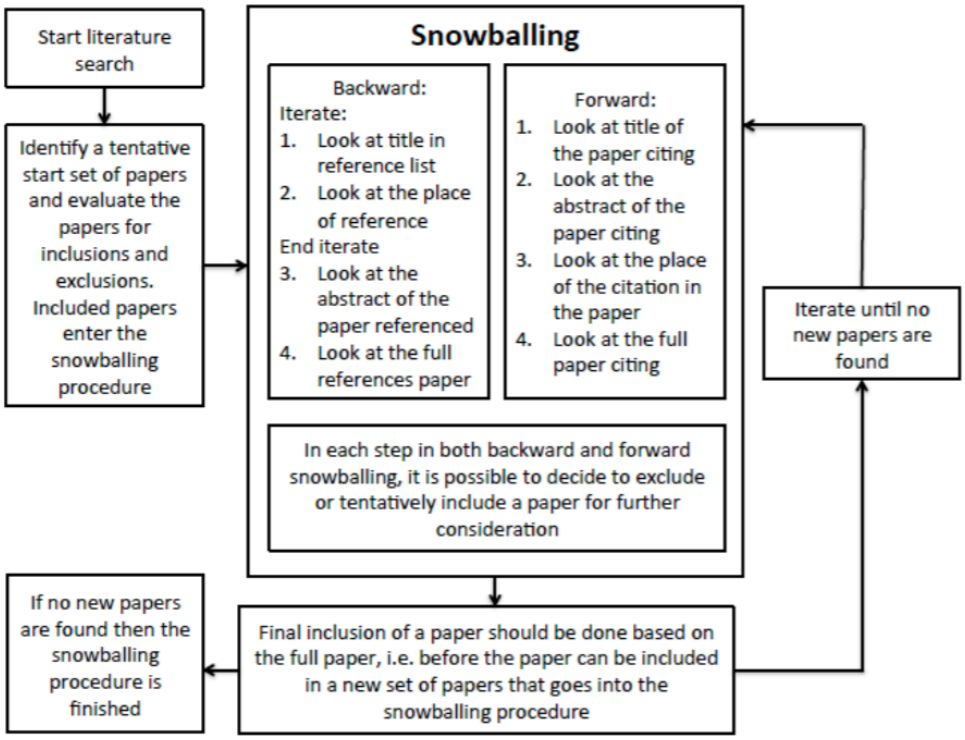

Figure 2.2: Snowballing procedure according to Wohlin (p. 4,[61])

Start set

Start set refers to identifying the base of the papers that will be used for the snowballing iterations. The formulated search string is inserted in the predefined databases in order to identify relevant papers for the review. This step is very challenging because the search goodness of the further papers is based on this start set. The start set presents the base of the next steps.

For this reason some criteria are defined to assure a reliable start set such as [61]:

- The number of papers in the start set should be moderate (not too small or too big).
- Selected papers in the start set should come from different communities.
- The start set should be based on the keywords and their synonyms.

Snowballing iteration

After establishing the start set, the snowballing iteration that includes backward and forward snowballing must be done. This step can be iterated many times until no new papers are found.

Backward snowballing:

Backward snowballing adopts the reference list to elect new papers. The first step is to go through the reference list and exclude papers that do not accomplish clear criteria such as time frame and language. Then the duplicates are eliminated from the list of the previous steps. After that, the inclusion criterion are applied to the papers through title, abstract and the whole document. First, the papers that are found through the reference list are evaluated over three steps. The title is examined at first. Inclusion criteria are applied on the title. The reviewer decides about the relevance of the candidate paper; if the paper is relevant, inclusion criteria should be applied on their abstract. If not the paper is discarded. In the next step, the evaluation is based on the whole content. In this way, the relevance and the efficiency of the papers will be guaranteed [61].

Forward snowballing:

It refers to identifying new papers based on those papers citing the article being examined. Each document citing the examined paper must be checked. The citations can be found in Google Scholar. Each candidate citing this studied paper are kept in this stage.

Likewise in the backward step, it is essential to decide for each new paper on either inclusion or exclusion. The snowballing iteration should be done

several times until no new relevant papers are detected and a rollback can be done in the two steps of snowballing procedure [61].

The snowballing procedure is very challenging and requires a lot of effort. Recent cases reported by Wohlin [61] also support that the efficiency of this research procedure is only 6.8%. For this reason, the author suggests some lessons to better perform the search with this method [61]:

- The frequency of papers identified in each step of the snowballing should be followed. If a good start set is defined, the number of new papers to be included will decrease after each iteration.

- When the number of the detected new relevant papers increases, the search should be executed once again taking into account synonyms. If the number of retrieved paper permanent increases, this means that there is a cluster of relevant papers missed due to ignoring some synonyms.

2.3 Collaboration in software engineering

As was pointed out in the chapter 1 of this thesis, software engineering is a collaborative process. Many programmers, software engineers, and architects coordinate their efforts to develop a large, complex and error-free software. The main drawback is that, most of the time, collaboration is a difficult task due to the ambiguity of the conversations between the team members and the limitation of the human brain to memorize every detail in the project. Due to these reasons, software collaboration techniques have evolved to address these constraints. Whitehead et al. [60] defined six main targets of this technique in his paper such as identifying, recording and resolving errors, recording organizational memory, and reducing dependencies among engineers. For more details, all goals and definitions of this technique are available in Whitehead's paper.

Another key term to define techniques to support the collaboration in the software engineering field especially in programming is "collaborative programming". This concept is a development approach that tends to be used to refer to the coordination of the individual programming tasks of programmers in order to perform a large and complex project [36]. Nowadays, collaborative software development is continuously expanding

[8]. The leading cause of this expansion is not only the augmentation of the number of outsourcing and offshoring projects. Team members of these projects are sometimes distributed over the globe, there are several advantages that team members can benefit from when they work in a team [8]. The key strengths of this method of work are the extended focusing on productive activities, the easy sharing of knowledge, the uninterrupted working sessions. Besides, the cost of the project is reduced, and the code quality is better [53].

Collaborative programming has many specifications. For example both novice and expert programmers can work together. Also, the team members can work into a distinct role, such as driver or navigator. This type of collaboration can be done, when they work with the same display, computer, mouse, and keyboard [11]. The driver uses the mouse and the keyboard to navigate and implement. The navigator, on the other hand, inspects for faults. Moreover, the collaboration can be distributed; that assures more flexibility to the developers. However, problem-solving in distributed collaboration is ambiguous despite the triviality of solving these issues in co-located teams [8].

Besides, the collaboration can be done in pair. Hannay et al. [22] conducted a meta-analysis on the effectiveness of pair programming in software engineering. The results of his study show that in low programming task complexity,one pair is agiler and faster than one programmer. Also the accuracy is better by complex programming tasks. In addition, very complex tasks can be accomplished by pair programmers but not by a single programmer [22].

2.4 Program comprehension

The expressions "software comprehension" and "program comprehension" are both used to express the same concept. According to Deimel et al. [15], software comprehension can be defined as follows: "the process of constructing the knowledge domains and relations among them from the code, comments, and whatever other documentation is available" . Besides Müller et al. [34] suggested to define it as: "building mental models of various abstraction levels, emerging from models of the code itself, to models of the underlying application domain, for maintenance, evolution, and re-

engineering purposes".

There has been numerous studies to investigate this research area due to the importance of program comprehension in the work process of the developer. It is the central activity of programming while it is a step to reach any goal namely fixing bug, maintenance or changing code etc. Program comprehension is essentially a cognitive process. Developers are continuously trying to understand an unfamiliar source code or to comprehend their code [51]. Currently the researches categorizes program comprehension into three main strategies:

Top-Down comprehension strategy:

This model is suitable for programmers that are familiar with a program's domain. The developer benefits from his know-how and his experience to suggest a hypothesis about the program goal and tries to compare the current code with others he knows [59].

Bottom-Up comprehension strategy:

This model is suitable for programmers that are unfamiliar with a program's domain or with the source code. The developers read the code carefully in detail and then group mentally the lines of code to build high-level abstractions. The aim is to affirm a hypothesis [51].

Integrated strategy:

The developer mixes both top-down and bottom-up models when one model is not sufficient [59].

Letovsky et al. [28] are convinced that developers are opportunistic so that they are able to take advantage on both top-down and bottom-up strategy. They suggest a comprehension model that is composed of three key elements as follows: knowledge base, assimilation process and mental model. O'Brien et al. [39] have updated the model of Letovsky et al. [28] and proposed a similar model but with four components. They added the external representation component. The figure 2.3 illustrates the proposed model by O'Brien et al.

It provides an overview of the key components of the understanding process. Firstly, "external representation" refers to external support that aid the developer by understanding the code. This support can be a tool, a

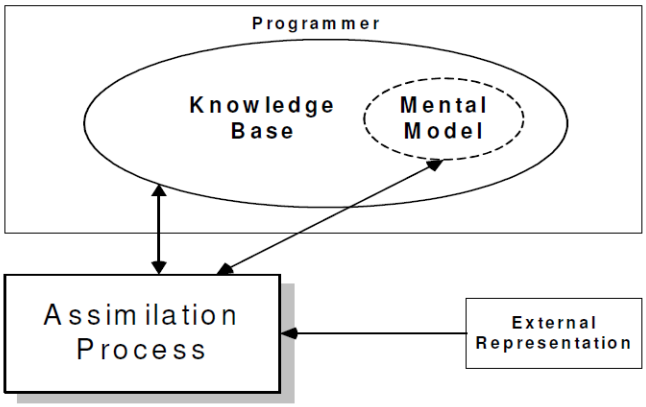

Figure 2.3: Components of software comprehension model according to O'Brien (p. 3,[39])

documentation, the source code itself or the advice from other programmers etc. Secondly, "knowledge base" represents the experience and knowledge of the developer before trying to understand the code. Thirdly, "mental model" refers to the instant representation of the developer. The mental model is continuously incremented while understanding. Finally, "assimilation process" represents the actual approach to comprehend the source code. Knowledge base and mental model are both used to achieve the assimilation process [39].

The process of program comprehension is a topic of interest. For more than 30 years, numerous studies have been conducted. The researchers invest a lot of effort and time to investigate how a developer comprehends a program. But how can the level of understanding be measured so that the effectiveness of the used strategy can be evaluated? To address this question, Siegmund [51] wrote a review that summarizes all the approaches that were applied in order to measure the software understanding over more than 30 years citation. The author notices that the most used approaches are think-aloud protocols, memorization and comprehension tasks. In light of the enhancement of the technologies, the diversity of programming tools and Integrated Development Environment (IDE), measuring the cognitive process during programming was very challenging. For this reason, another new approaches have been applied such as functional Magnetic Resonance Imaging (fMRI) and EEG signal.

Siegmund [51] is not only interested on the past and present approaches

but predicted also the tasks and the future approaches that will be used. The author anticipates that a new model of program comprehension will be conceived. The source-code level will not be the centre of interest but the software comprehension tasks of the future will be included in the overview of the complete program, the hierarchy between the structure, the correlation between the components and the task activity of each developer in the team. She was right in her prediction because as pointed out in the motivation, the goal of this review is to summarize the approaches that are used to steer the attention of the developer in order to maintain an overview of the large software and to enhance the awareness inside the team [51].

Chapter 3

Related work

This thesis aims to review papers about steering the attention of the developer during collaborative programming in order to maintain an overview of the important parts in the document or the code and to enhance the program comprehension. In the software engineering field, there is no systematic literature about this theme. For this reason, this chapter will combine the done systematic literature reviews about tools that supports program comprehension, software visualization tools and eye tracking methods that are considered to maintain the overview.

The first section will include research area from program comprehension, the second section will encompass the works about software visualization and the third section will inhere systematic literature review about eye tracking methods.

3.1 Program comprehension

Program comprehension is not a goal in achieving but a necessary step to achieve some other aims like fixing errors, maintenance, reusing code, refactoring, testing, analysis of changed or functionality change of a program, etc. Storey [55] provides an excellent study inspired by a literature review about theories, tools and research methods in program comprehension over the past thirty years and gives predictions for future works. She covered the fundamental cognitive theories of program comprehension and used it to examine the tools that support program comprehension. The correlation between the performance of the tools and the theories is also studied to

assess the available tools. Classification of both theories and tools are also done conforming to the context, type of the program and individual aspect. Finally, a prediction for future work is made [55].

The tools that support program comprehension are of significant interest within this review. Storey [55] summarized the requirement that should be present to support the program comprehension according to the recommendation of researchers after performing studies with expert programmers in an industrial environment. Different researchers recommend concept assignment problem, reverse engineering tool needs, the importance of search and history, information needs for maintainers and software visualization tool needs, all of these features. The replacement of one feature or the addition of another is not a trivial task. For this reason, each feature will be briefly defined and each programmer should choose the appropriate features according to his needs.

After explaining the feature requirement, Storey [55] categorized the programmer (distributed team, agile developer, etc.) and program characteristics (distributed application, diverse source of information, etc.). Storey has not only summarized the past and present theories, but also she gives guidance to develop the tools and the theories in the future. She criticizes the available research methods and gives some advice on how to address the issues. For example, she evoked that experiments in an industrial setting afford divers organizational challenges like the massive amount of information provided by observations, make the data analyzes problematic. Moreover, the Hawthorne effect is unavoidable. Due to these reasons, Storey suggests using benchmarks and collaborative tool demonstrations in order to compare their tools with others and understand the gaps and similarities between the approaches. She emphasizes that the goal is to understand why the tool is better than the others in the aim of enhancing the tools. In the other hand, she is convinced to ameliorate the theories by providing the documentation and results to other researchers so that they can understand the data and the conclusions [55].

Last but not least, she predicted that the field of program comprehension would incur more importance due to the enhanced technologies as well different researchers from different domains becoming interested in recognizing the cognitive and social aspects of this research area [55].

Despite Storey [55], Schröter et al. [47] conducted a systematic literature

review about understanding studies about program comprehension . The central question in this dissertation asks "how researchers address program comprehension" according to the context, terminology, and threats to validity. The authors studied 540 research papers published at ICPC between 2006 and 2016. Before answering the first question, the authors categorized the program comprehension into seven parts (i.e., source code, program behaviour, testing, API, requirements, documentation, and miscellaneous). This systematic literature review aims to give an overview of the progress of the published papers over time and to evaluate the specification of the author about the research area and their assessment. According to this review, source code and program behaviour are the two parts of most interest. Moreover, the ambiguous terminology makes the search and the comparisons of the study more difficult. Besides, the evaluation of the threats to validity proves that newly quality of papers is enhanced [47].

3.2 Software visualization

The focus of software visualization area is to visualize the structure, behaviour, and evolution of software. For Mattila et al. [33], software visualization refers to "visualizing various aspects and artifacts related to software". From this definition, various aspects of software engineering can be interpellated for instance program comprehension and comprehension of a software process. She conducted a systematic literature review of the research papers that were presented from 2010 to 2015. The scope of this survey is to address the following two perspectives about the focus and the maturity of the software visualization field. 83 papers are included in this survey. According to the results of this survey, different tasks are supported by software visualization such as program comprehension, collaboration, and engagement, maintenance, etc.

Moreover, software visualization deals with change over time for instance structure of the software, working habits, performance, dependencies, states of project, resource usage, time, software product line variants, user's activities, data flow, and workflow. Besides, hierarchical or graph visualizations are the most used visualization methods; this result can be explained through the main task of a visualization tool that is understanding the structure and the behaviour of the code. Geometric projection techniques

like polymetric views, city metaphor, and tag clouds are also often applied in the visualization. Another important finding is that generally source code and software execution data are employed as a source for data visualization tools. As regards to the maturity of software visualization, Mattila et al. noticed that 90% of the papers discuss new visualization tools and 10% of the rest focuses on the evaluation of the method or using existing methods in the new research area. Interestingly, she observed that only 20% of the papers include the research questions directly and 67.5% assessed their new tool or method [33].

In contrast to Mattila et al. [33], Seriai et al. [48] conducted a systematic literature review on the techniques that are applied to validate software visualization tools. 87 research papers which were published between 2000 and 2012 are included in the survey. The result of the study shows that 72.5% of the research papers include a precise assignment; 77% of the studies use open source projects data to evaluate their tool. 60.9% of the researchers employ objective measures for example time task completion, and the number of failures. Surprisingly, 70.1% of the studies are without participants; the tools are only compared directly with other tools [48].

3.3 Eye tracking

Eye tracking is a new data collection method in the software engineering field. The collected data is the visual attention of the user. It can be displayed and used for different tasks.

Sharafi et al. [49] performed a systematic literature review that covers the usage of eye tracking in software engineering between 1990 and 2014. The survey includes 36 publications. They provide many details on the history of eye-tracking and the various setup of devices. Henceforth, the researchers apply eye trackers to examine code comprehension, debugging, collaborative interaction, model comprehension and traceability. Furthermore, different metrics based on the eye movement are employed to assess the visual effort of the participants and highlight the way of detecting the stimuli. The disadvantages of eye tracking are also determined. Interestingly, Sharafi et al. [49] noticed that the researchers use SLR despite the drawbacks that this method had.

Complementary to Sharafi et al. [49], Obaidellah et al. [38] report the

results on the usage of eye-tracking in computer programming [38]. He analyzes the results of the studies in relation to the experimental setup. In this review, types of participants, and trackers are taken into account. He includes the publications that are published between 1990 and 2017. A total of 63 studies are used. Sharafi et al. [49], Obaidellah et al. [38] use the same categorization of programming tasks. Obaidellah et al. [38] identify that nowadays program comprehension and debugging reach a higher level of interest, in contrast to non-code comprehension, collaborative programming, and traceability. Researchers, students as well as faculty members take part on the most studies. Also, they report that the most used tool to track the attention is eye-tracker Tobii [38].

Chapter 4

Review method

This chapter describes the steps used in this literature review that is based on the guidelines of a systematic literature review by Kitchenham [26]. The first section defines the research question and search string. The second part presents the inclusion and exclusion criteria. The third part discusses the method of selection of the papers. The fourth part gives an overview of the retrieved results. The fifth part moves on to analyze the content of the findings in greater detail .

4.1 Research question and search String

The definition of the research question is fundamental in the literature review. It helps the author to focus on the important topic of the research and it is also crucial for the reader, since it outlines the literature that will be examined in this review. This research seeks to address the following questions:

RQ 1:

What are the approaches that are used to steer the attention of the developer and to support him to have an overview on the whole project or to understand a part of a document?

RQ 2:

What are the different techniques that are used to measure the attention of the developer during program comprehension?

RQ 3:

How is the interplay between collaboration and the attention of the developer investigated?

In table 4.1, the research question will be briefly explained as follows:

Table 4.1: Target of the research questions

Research question	**Target**
RQ 1	Determine which approaches are used to steer the attention, to support the developer and to maintain the overview of the project and how they can be used.
RQ 2	Identify and analyze the techniques that are used to measure the attention of the developer.
RQ 3	Determine the correlation between the attention of the developer and the collaboration level.

The table 4.1 includes the research questions and the focus of these questions. The objective of the three questions is to recognize the approaches and the techniques that are used to steer the attention and to support the understanding of the developer during the software development. Furthermore, to measure their attention during this process, and after that to detect the relation between the collaboration level and the attention of the developer.

After defining the research questions, the next step will be the definition of the keywords. They are used to highlight the important words to answer the research questions by forming the search string. Similar terms of the main concepts should be also taken into account. The keywords are formulated as follows: attention, collaboration, cooperation, teamwork, program comprehension, program understanding, program debugging and software development. From these keywords, a search string is created to address the research question.

Search string:

```
("program comprehension" OR "program understanding" OR "program debugging")
AND (collaboration OR teamwork OR cooperation)
AND ("software engineering" OR "software development")
AND attention
NOT education
NOT network
```

The search string above consists on the combination of the previously defined keywords. The terms "program comprehension" OR "program understanding" OR "program debugging" are fundamental in the case of this study because the target is tracking the behaviour of the developer until the program comprehension process. The words "understanding" and "comprehension" are used synonymously to reach the highest number of relevant papers but "debugging" is used to reach the paper that seeks to address program debugging. The Boolean operator "OR" is used because a minimum of one of this terms is needed. "Software engineering" is the field of the study, it is used to limit the field of the search and "Software development" is inserted to insist on the process of the development. "attention" is the most important word in the review. It is the basis of the theme because the aim is to steer the attention of the developer during software development in teamwork and how this information can influence the efficiency and the accuracy of the collaborative work. It is used to pick all the papers that are related to the attention of the developer. "Collaboration", "cooperation" and "teamwork" are synonyms which explain why an OR operator is put between each word. These three keywords are crucial because the attention should be evaluated in a collaborative environment. All these keywords are connected with AND operator because all these keywords should be included in a document to be more likely to attain the relevant paper. The NOT operator is also used to eliminate some topics that should not be included. For example: education and network. The majority of papers that include the word "education" are also excluded as this word represents "the eLearning" field and education in a classroom. In the end, the word "network" is discarded because it includes the domain

of the network and network security.

4.2 Inclusion and exclusion criteria

The exclusion and inclusion criteria in table 4.2 are adopted to elect the relevant papers to review.

Table 4.2: Inclusion and exclusion criteria applied to the review

	Criteria	**Desription**
Inclusion	IC1	The paper presents an experimentation or an empirical study.
	IC2	The paper is a descriptive analysis.
	IC3	The paper is a peer-reviewed contribution to a conference or a journal.
	IC4	Papers published between 2003 and 2018.
Exclusion	EC1	The paper has no accordance with at least three of the search keywords.
	EC2	Focus on the difference between male or female and disabled and not disabled persons.
	EC3	Paper is not accessible.
	EC4	The Paper is duplicate.
	EC5	Paper is not in English.

As shown in table 4.2, the criteria that are used, are divided into inclusion and exclusion. Inclusion criteria are defined in advance to identify subjects which will be included in a research study.

IC1: The paper presents an experimentation or empirical study in program comprehension, this inclusion criteria is very important for the review because experimentation or an empirical study must be done to orient the attention of the developer during the program comprehension.

IC2: The paper is a descriptive analysis of program understanding. This inclusion criterion is crucial because the analysis should be inside the team that will participate in the experimentation. A summary of the behaviour of each developer must be done to make conclusions.

IC3: The paper should be peer reviewed because this is a literature review and only peer-reviewed articles should be selected.

IC4: The papers that are published between 2003 and 2018 are included due to the rapid development of the technology since 2003.

Now, the exclusion criteria will be discussed. This criterion is also determined forward to elect the peer-reviewed papers that responds to the research questions.

EC1: The paper has no accordance with at least three of the search keywords. This exclusion criteria is decisive under the importance of the keywords that are included in the search string.

EC2: The experiment should not focus on the difference between male or female and disabled and not disabled persons. The aim of this review is to take general conclusions.

EC3, **EC4**: Not accessible and duplicate papers are an exclusion criterion. It is evident that duplicated or not accessible papers cannot be used.

EC5: Paper is not in English. This exclusion criteria is adopted because this review is designed to the international researcher.

4.3 Method of selection of the primary studies

After defining the search string and the inclusion and exclusion criteria. The search string is inserted in five databases such IEEExplore ,SpringerLink ,ACM digital library ,Science Direct ,and Google Scholar. These databases are the most relevant on the software engineering field except for Google Scholar is a multi-disciplinary database. Five databases are used in order to seek the most possible relevant paper.

In this review, the selection of the studies took place in six steps. Firstly, time range and language criteria are defined to decrease the number of the outputs in order to facilitate the search on the relevant paper. Secondly, the selection of primary studies is based on the title. If the decision based on the title is ambiguous, the abstract should be read to decide if the article should be included in the start set or not. Thirdly, the decision making is based on the abstract. The reviewer should read the abstract then decide about the relevance of the paper. If the article has remained, the fourth step should be achieved. The reviewer should read the whole paper, then judge the importance of the paper. The selected papers are considered as the start set. In the final step, forward and backward snowballing must be done as pointed out in the chapter 2. This step must be done until no relevant paper is found.

4.4 Results of the Selection

After inserting this search string in the five databases and using the exclusion criteria of language and the time range of the publications, 140 papers in IEEExplore, eleven papers in SpringerLink , 44 papers in ACM digital library, 75 papers in Science Direct and 538 results (books, dissertations and websites) in Google Scholar have been found. The search was done on 22.12.2018. The second step is to apply the inclusion and exclusion criteria to the title, all the claims of papers that may not be included in our review must be excluded. After this step, 20 documents in IEEExplore, three articles in SpringerLink, eight papers in ACM digital library, ten papers in Science Direct and 31 papers in Google Scholar are taken to the third step. 71 papers fits the scope of the research based on the title. The fourth step is eliminate all the duplicates, so 62 papers are remaining for the next step. Now the inclusion and exclusion must be applied to the abstract, so 13 papers remain in the set. After reading all papers, six of the papers are retained in the start set. In the next step, the snowballing search procedure is applied. The backward and forward steps of snowballing are repeated five times until no new relevant paper is found.

The table 4.3 reports the results of each iteration after applying the inclusion and exclusion criteria based on the title, abstract and whole article.

Table 4.3: Snowballing iteration results after the inclusion and exclusion criteria based on title, abstract and the whole text

Iteration number	Backward	Forward
1	3	5
2	4	4
3	1	2
4	1	0
5	0	0

The table 4.3 above illustrates the results of the forward and backward steps of the Snowballing procedure after each iteration. Five iterations are needed until no new relevant paper is found. Each document is evaluated based on the inclusion and exclusion criteria and eliminating all duplicates.

In the backward snowballing, the references of the included papers are studied to select more documents to add in the review. In the forward

snowballing, the papers citing the papers in the start set are assessed. The time range and the language are also taken into account. At this moment, the snowballing is completed, but a roll-back can be done at any time. In the following step of the review, 26 papers will be considered in more detail. The following diagram reports the literature search strategy with the snowballing technique.

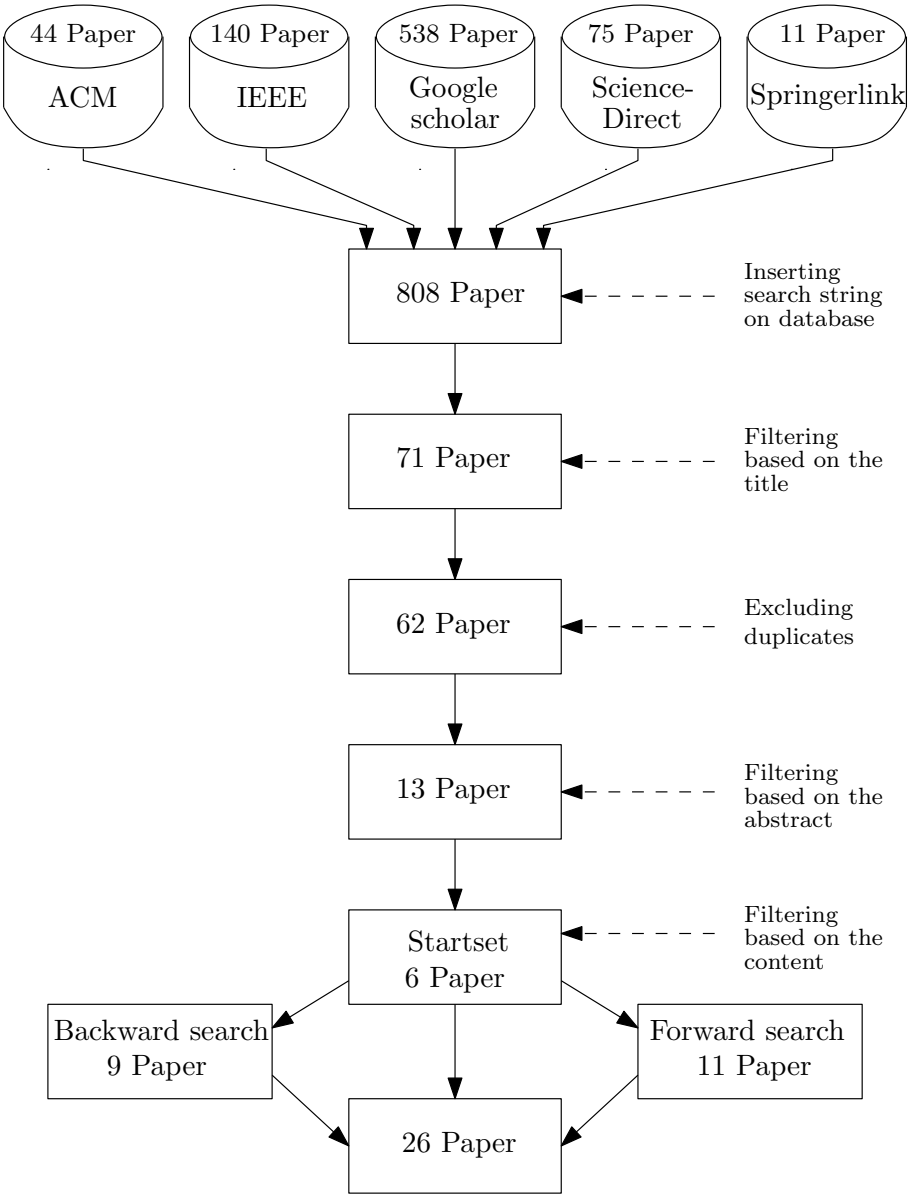

Figure 4.1: Search with snowballing procedure end results

From the figure 4.1, an overview of the discovered papers during the process of the search and how this number was developed, is shown. After applying the criteria of the time range, language, type of document and based on the title, 71 papers were identified. After that all duplicates are eliminated, 62 papers remain in the set. Subsequently, it is decided whether or not the article should be included based on the abstract. 13 papers have remained. In the next step, the papers are filtered with respect to the inclusion and exclusion criteria based on whole text, only six papers stay in the set. The snowballing technique is applied five times until no new article is found. In the backward snowballing, eight papers are selected after considering the inclusion and exclusion criteria based on the title, abstract and the whole text. As in forward snowballing, nine papers remain in the set. In total 26 articles will be analyzed in the next step.

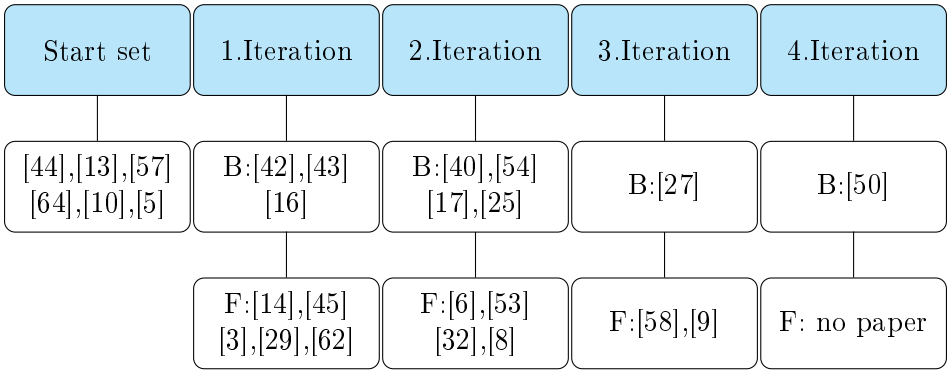

Figure 4.2: Overview of the retrieved papers of each iteration

The figure 4.2 illustrates the selected papers of the four iterations of the snowballing procedure. The result of each iteration is divided into two parts "B" means backward snowballing and "F" means forward snowballing.

4.5 Analysis of the papers

In this section, each paper of the set will be analyzed. This section is divided into five parts. The first subsection includes the papers of the start set, from the second to the fifth subsection the outputs of each iteration are included. The papers of each part are named from Iij such that i denotes the number of the iteration and j indicates the number of the identified paper in this iteration and for the paper of the start set i is equal to zero.

4.5.1 Start Set paper

I01. Pietinen et al. [44] conduct an empirical study with a duration of two months in industrial-like settings. The central point of this paper is the relation between the eye movement of the developers during pair programming and the efficiency of their work. Eye tracking and verbal protocols are used to record shared visual attention during collaborative programming to propose novel eye-tracking metrics. The eye movement and verbal protocols of two participants are simultaneously recorded for the study. Only the initial results of this study are reported. A visualization of the eye movement is displayed to find associations and conclusions about the metrics that can be used for the next researches. The results obtained from the observation of the eye movement indicate that the number of overlapping fixation gives an account on the type of collaboration. For example, a high rate of overlapping fixation is probably caused by a great collaboration, high fixation duration on the overlapping fixation and also long gaze duration. Therefore, this means that the developer has a comprehension problem. This hypothesis will be tested in further research by Pietinen et al. [44].

I02. Cook et al. [13] report an empirical assessment of Collaborative Software Engineering tools. The hypothesis of the author is as follows: employing collaborative software engineering tools has a lot of advantages for example in the task time completion. Twelve developers work in pairs to complete a specific tasks. The participants work in two modes: conventional and collaborative. Collaborative Architecture for Iterative Software Engineering (CAISE) based Collaborative Software Engineering (CSE) tool is used by each pair. The participants should modify the present code. In conventional mode, the version control system alerts the user on changes. However, in a collaborative way, the user can see the modification done by the other users, therefore, avoiding conflicts. Task completion is used to assess the effectiveness of the use of CAISE in collaborative programming. One way ANOVA is employed to analyze the measurement. The obtained results prove that the task completion time in collaborative mode is significantly less than by conventional mode. A survey is also used to analyze user preferences. The results of the survey accord with the earlier findings, the users appreciate these tools [13].

I03. Sulir et al. [57] pilot two controlled experiments. In particular, this paper will examine three main research questions: "1. Do programmers'

mental model overlap? 2. How do developers use shared concern annotations when they are available? 3. Does using annotations created by others improve program comprehension and maintenance correctness, time and confidence?". The first experiment focused on the effect of annotations in program comprehension and maintenance by students. It showed that there is an overlap between mental models and used concern annotations. Consequently, they can be shared. The statistical finding of the first study highlights the improvement of the development time by program comprehension and maintenance tasks. The second experiment is nearby the same as the first one, but it focuses on the industrial developers. This experiment aims to study the correctness of the code when developers use annotated code, although the results differ slightly from those of the first experiment. The time of development is slightly worse, but the task and questions correctness is ameliorated. This results proved that the shared annotations help the developers to enhance their program comprehension especially by feature location and confirmation of hypotheses and gaining new information [57].

I04. Chu et al. [10] represent an observational study on pair programming with two participants. This study is guided on pair program comprehension and employs the examined results to create a tool that contributes to collaborative awareness of a programmer. This research seeks to address the following hypotheses: semi-structured goal-question evidence methodology for program comprehension should be used to preserve an overview of accessible documentation by the team member of a project. This approach should have three aspects: re-documentation system should work in parallel while programmers were developing, the knowledge and objective of team should be dispersed in the group, and occurred problem should be known by all member of the team. In order to prove these hypotheses, an observational study is undertaken. The participants are observed while they work in pairs with a digital video camera. Sound and image are recorded for each session. The findings of this study (observed scenarios, event magnitudes, and event relationships) are employed to develop a better cognitive support tool for the program comprehension. The proposed prototype "Pollinator" is an eclipse plug-in. The main functions of this prototype are improving the collaborative programming and contributing three views classified by awareness, knowledge ,and comprehension building

view. Program understanding project and user awareness information are encompassed in awareness view. The target of program comprehension are included in knowledge base awareness view. An hierarchical diagram of comprehension can be performed among the comprehension view. The findings of this study confirmed that the research is on the right path and the research will be completed [10].

I05. Zayour et al. [64] convey a qualitative study on debugging under an enterprise IDE, in a real industrial setting. 17 programmers are observed while dealing with bugs for 117 hours. Afterwards, they are questioned about their resolution to solve the bugs, the cause of the difficulties after executing multiple tasks. The goal of this experiment is to recognize the problems that are related to the debugging under enterprise IDE. The results of this study revealed that most debugging activities are associated with the type of enterprise IDE. The most obvious finding to emerge from this study is that enterprise IDE platform enhances the way of work by developers. Thus it organizes the work of the developer and lets him profit from the expertise of the other programmers. The participants recommend some optimization such as a better debugging and better reporting tool [64].

I06. Biehl et al. [5] report a field study about a new interactive visualization technique Fostering Awarness for Software Teams Dashboard (FASTDash).The target of this tool is to enhance the collaborative work in software development for example by avoiding conflict situation. FASTDash displays the current activities of the developers using a dashboard or the own display of each developer, for spatial representation of the shared code base. To evaluate this tool, a survey, interviews and in situ observations are done. Six experienced programmers participate in this observational study. The participants are located in the same room. To categorize user behaviour, new coding schemes are generated. Pre-FASTDash and Post-FASTDash observations are achieved to deduce the effect of FASTDash on the teamwork. The results of this study indicate that most of the participants use the information about who is changing, what is changing, which file is checked out and error report to enhance their awareness while working. Also, they use a different source of information such as whiteboard, notes, bug databases, and source code. But methods to improve the awareness depend on the person. For example, participants that are accustomed to agile methodology prefer verbal communication. Contrary to expectations,

the use of the broad share dashboard impedes the work of the team for many reasons. One main drawback is that the large shared workspace displays only information about one developer at a time [5].

4.5.2 First iteration

I11. D'Angelo et al. [14] conduct a survey, pilot observation, study of remote pair programming task ,and post-task questionnaire. Three teams take part in this study. Video recorded observations and notes by the researchers are used to analyze the approaches that are used in pair programming. The three teams executed different practices of pair programming. The target is to analyze the eye gaze and video of the participant to enhance the collaboration in the group. It uses the eye tracking and gaze awareness method to help pairs to see what the other person is looking at in a code document. This novel tool aims to improve the developer's ability to communicate about on-screen locations efficiently. This new approach allows the user to know where his partner is looking and to change the colour when the pair is looking at the same position on the screen. It is interesting to note that in most cases of this study, the pair spends more time looking at the corresponding locations, communication about reference point become faster and more successful [14].

I12. Roehm et al. [45] describe an observational study of 28 professional developers from seven companies. An observation with the think-aloud method and an interview are performed based on the program comprehension of the programmer in industry. The purpose of this study is to follow the strategies that are used, the information that is needed, the tools that are used by the developer to comprehend a program and to test the validity of hypotheses that are related to the program comprehension in industry. The most striking result to emerge from the interview and observations is that the programmers try themselves to understand the program instead of the final user , and sometimes they clone the code to avoid program comprehension. Another observation is that the developer prefers to communicate within a team face to face rather than through documentation and the experience of the developer equips him the understanding of the program. In contrast to the findings of state of the art, program comprehension tools are unknown or infrequently employed by the developer. This experiment confirms the existence of gaps between research and practice in program comprehension field [45].

I13. Baltes et al. [3] conduct a qualitative user study. Twelve developers participate in pairs in this study. They work in two open source projects in a controlled setting. The executed task consist of debugging a code then answering a questionnaire. The target is to see how the developers detect the bug and how they communicate the issues in a team and which strategies are used to debug. To track the participants, profiling tools are used to measure individual program runs and to display profiling information in the source code view. This study is aimed at addressing the following research questions: "1. how do developers navigate and what information and representation is supportive for locating a performance bug? 2. How do developers try to understand and explain the causes of performance bugs?". On the first research question, this study found that two navigation strategies are recognized. The first strategy is alternating between testing and coding. The second strategy is tracking the path through the dynamic call graph. Another finding is that dynamic instances (calls as links) and time consumption are very relevant to identify a bug. And adding these two pieces of information in the code will be very helpful in enhancing the work of the developer. In the second research question, this study has shown that hypotheses and sketches are defined to help the team to detect the reason for the bug and to explain it to each other [3].

I14. Pietinen et al. [42] introduce a framework that studies the visual attention using eye-tracking during pair programming in the real world. In this paper, challenges and requirements on the eye-tracking setup are discussed as well as some software problems and solutions when eye-tracking is used in pair programming. Two developers work in a pair in a distributed environment. The eye movements of both programmers are recorded and synchronized together. The first problem that occurs is the seat position of the participant. It is not optimal for the eye tracker also the body movements are not considered in this research although it is critical. The analysis of eye gaze data is manual, due to the difficulty of the analysis as the video that contains both developers eye-movements and the scene are on a single screen. The second problem is the synchronization of the eye movements in the same video. The solutions to these problems are explained in this study. In conclusion, this paper seems to improve collaborative visual attention analysis. Another finding is that the available eye-tracking system does not satisfy the requirement of the actual research, other systems should be added

to reach good data quality[42].

I15. Maalej et al. [29] pilot an observational study, and an online survey. 28 developers take part in this study. They ask five main research questions as follows: "1. which strategies do developers follow to comprehend programs? 2. which tools do developers use when understanding programs and how? 3. which knowledge is important for developers during comprehension tasks? 4. which channels do developers prefer to access and share knowledge about software? 5. which problems are frequently encountered by developers while exchanging knowledge about software?". Think-aloud method and observation protocol are used to record the actions of each participant so that answers about the research questions can be found. The results of this study show that there is a gap between research and practice, for example, program comprehension tools are not used in the industry, besides researcher and professionals see program comprehension differently. These differences can be explained by the lack of knowledge or trust in the new program comprehension tools. Further reported finding shows that the strategy that is used by the developer to comprehend a program depends on the task. As no strategy fits all the tasks, the author suggests integrating a context-aware tool to support the knowledge exchange and to detect only the relevant information. The developer prefers face to face communication rather than documentation, and their experience is very crucial to comprehend the program. Furthermore, the channel that is used to communicate knowledge is different between the knowledge seeker and knowledge provider. The seeker prefers project documents, familiar team-mate but the provider prefers comments, notepads, etc. Also, the problem that is detected inside the industry is that the documentation is obsolete. Finally, there is a correlation between team size, previous open source experience and the need for knowledge [29].

I16. Ye et al. [62] represent a newly developed tool Team-based Workspace Awarness Toolkit and Collaboration Hub (TeamWATCH). The purpose of this is to visualize developer activities using a 3-D city metaphor to enhance the collaboration inside the team by maintaining awareness. This tool allows the extraction of awareness information from the control repository, local workspaces, and tracking system. Real-time and historical data are also displayed. TeamWATCH permits the developer to choose between two modes of visualization (overview of the team activities or

individual activities). To test the efficiency of this tool, the following research question is asked: "do the developers who use TeamWATCH detect and resolve potential conflicts earlier with lower merge conflicts, compared with the developers who do not use TeamWATCH ?" A user study is done to answer the previous question. The experimental data is collected from different sources (video recording, chat logs, Concurrent Versioning System (CVS) repository, chat logs, and survey questions). Two groups of participants are asked to accomplish the same five tasks (one group use TeamWATCH and the other one not) and to answer a survey. After that, the collected data is statistically compared. The results significantly shown that TeamWATCH helps the user to detect and solve conflicts earlier and better. Moreover, Ye et al. prove that the collaboration is increased [62].

I17. Pietinen et al. [43] conduct a controlled study that is done in an industrial-like setting for two months. Students participate in this study. The data is collected during a project called "software productivity". Eye tracking is used to track two developers visual attention simultaneously during program development. They are not only interested in the recorded eye-movements but also on the psychology of programming of each programmer. The results cast a new light on the importance of the protocol, the division of roles, avoiding free driving, information search, and elaboration on findings. The results of this study indicate that for studying pair programming practices, it is essential to continuously review the experiment to know which task is relevant and which not and to partially use pair programming. Also the findings of this study suggest that pair programming can be replaced with side-by-side programming that might have a lot of advantages like better knowledge management, enhancement of the communication and the best method to study pair programming is the use of eye-tracking instead of think this avoids the problem of interruptions during the accomplishment of the tasks. It's better to use it in conjunction with other protocols to have more level of detail [43].

I18. DeLine et al. [16] convey a laboratory and a field study. In the first study, nine software developers take part. The participants seek a precise task, one group with the help of "team tracks" tool and the other without any help. Henceforth the task completion is compared, and the users are questioned about their satisfaction after using "team tracks". In the second study, the participants work in teams. Five developers work in a group where

they are asked to use "team tracks" to solve a specific problem. After that, they are asked to rate this tool. The goal of the development of "team tracks" is to address the following issues that are detected by the professionals. Firstly, the lack of documentation makes the identification of the relevant part of the code or the relation between two parts of the code confusing. Secondly, there is a loss of the overview when a lot of documents are open. Thirdly, the textual search function is deficient. To address these problems, "team tracks" is developed. This tool guides the attention of the developer on the relevant information and helps him to navigate in the code without loss of path by displaying the source code navigation and other information to the development team. The first user study found that "team tracks" is easy to learn and to use and facilitate the comprehension of the task. The results of the second study indicate that "team tracks" reduce the cognitive and memory load of the developer [16].

4.5.3 Second iteration

I21. Omoronyia et al. [40] conduct an empirical study. The authors suggest a tool that enhances the shared awareness during programming tasks by automatically recording developer IDE interactions. This interactions can be real-time executed actions or tasks. In this paper, the team can also be distributed geographically. Ten advanced software engineering students take part in the evaluation of this tool Continuum of Relevance Index (CRI) model. They are working in a group of three. The collected data through CRI, audio record and an interview after achieving the tasks help the author to analyze the efficiency of this new tool. The results of this study show that it is possible to create a tool that captures all the interactions of the user in a team. Another important finding is the role of social graph view in providing an accurate summary of the events of the collaborative projects. Last but not least, the CRI model increases the awareness for example by the function "slide through" enables the user to scroll on the history of the project thus they get a complete overview of all the activities that are executed along with the project. Based on the analyzed data, the author concludes that it is possible to create a tool that captures all the interactions of the user in a team to enhance the awareness [40].

I22. Busechian et al. [6] suggest a new generation tool, wireless full channel EEG device. This tool allows one to perceive in depth the mental

processes present in developers during phases of development and how this process varies when different development approaches are used. The purpose of this work is to detect the most effective conditions, procedures and practices to enhance the quality of software systems with high efficiency. The metrics that are measured are task engagement and mental workload. Electrodes are placed all over the head of the four participants to obtain a brain map with different brain areas. Four subject works in pairs to prove that pair programming procures a higher level of concentration. The goal of this experiment is first to see the applicability of this solution then to analyze the collected data. Three waves are considered in the interpretation such as alpha, beta, and theta. Alpha waves permits one to conclude that while pair programming developers are accomplishing the tasks in a relaxed mode, beta waves revealed that tasks that required concentration to solve a problem are present in solo and pair programming but the coloured region in the brain is nearly the same, but in a pair is less intense. Moreover, last but not least theta waves showed that solo developers desire to eliminate distractive stimulus in order to stay concentrated on a task. From this short experimentation, the do-ability of this new approach is approved, and the results demonstrate that there is a difference in brain activity between pair and solo programming. Busechian et al. [6] confirm that EEG can be adopted to evaluate the effectiveness of processes and practices in software engineering.

I23. Stein et al. [54] aim to address the following hypothesis: seeing the visual attention of a developer may help other persons to accomplish the same assignment. A small experiment is done in order to prove or to disapprove this hypothesis. Professional programmers participate in this study. They are asked to wear a head-mounted eye tracker and to solve a specific task. Their actions and speeches are also recorded. In the second step of the study, another group of programmers should see the eye gaze of the first group and should solve the same tasks. The current study found that there are two manners to detect a bug with the help of eye gaze. It can be done through identifying where the edit point for the failure is or through delimiting the part where the cursor persistently alternates. Besides, the results of this experiment prove the accuracy of the hypothesis of the author. However, the user should be able to remember on the eye gaze trace in order to use it later. A significant limitation needs to be considered. Eye gaze may be

confusing if it does not trace a simple path [54].

I24. Chen et al. [8] present a new visual mobile approach to maintain a continuous awareness for distributed teams. It is devoted to managers and developers. It consists of using mobile devices when the user is out of office, but the desktop version is also available for users inside the office. The user can choose between an online or an offline mode. Online mode assures an up to date awareness information. Push up notifications guide the attention of the programmer on the important up-date or cooperation moment. Information awareness is spread in "Team Radar" by catching, spreading, analyzing and visualizing the interaction data of interest. To evaluate this new tool, the author tries to address this following research questions: "1. does the visualization on "Team Radar Mobile" increase the correctness of the answers to the awareness perception questions, compared to non-visual approaches? 2. does the visualization on Team Radar mobile reduce the time needed for the awareness perception tasks, compared to non-visual approaches?" An experiment is done to answer this question. 14 participants take part in this study. They are divided into two groups. Each group is asked to execute some tasks. One with the help of "Team Radar" and one without "Team Radar". The evaluation is based on the time completion and the average correctness of the task. The study investigates the capabilities of "Team Radar" to detect conflicts, work dependency, project evolution, expert location, and developer activity. The results of this study show a significant improvement of the correctness and completion time for program comprehension activities. In this case, the visual approach exceeds the non-visual approach. However, for scanning the script, the visual approach did not improve the correctness but enhanced the time to finish the tasks [8].

I25. DeLine et al. [17] report the results of a formative observational study. Seven developers perform this experiment. The main task is to modify an unknown code of a popular video game. A major problem with this kind of system is the deficient overview of documentation so that the navigation becomes very difficult. Computational wear with social filtering is developed to address these problems. To collect the data about the process of program comprehension of the user, custom designed logger was used to record the code modification. Also, a think-aloud study is done. The main aim of this investigation is to direct the attention of the developer to the position

where most of the programmers have performed through three conceptual visualizations using wear-based filtering. The first visualization is "FAN List"; it helps the programmer to find a given definition without changing the current focus quickly. The second visualization is "Code Favorites"; it supports the developer to steer his attention to the important parts where his team members have worked most of the time. The third visualization consists of "Wear for degree-of-interest". It is used to support the developer to have a better overview of the system components and their relationships among an automatic generated Unified Modeling Language (UML) diagram based on the interaction history. This diagram displays the whole project and the level of activity of each previous programmer in each part. This third visualization helps the developer to maintain the overview [17].

I26. Sillitti et al. [53] present a large case study in an industrial environment. 17 developers take part in this study for ten months. This paper investigates how pair programming influences the kind of writing and interacting with the computer of the user. It focuses especially on the comprehension, the attention and the productivity of the developer. To collect the data, PRO Metrics (PROM) is used to have a complete overview of the development process of each developer. For the analysis of the collected data, three techniques (L-graphs, Cycles, and sequences) are used. This experiment has conclusively shown that pair programming encourages the programmer to focus on their work so that their productivity is enhanced. L-graphs have demonstrated that developers were more effective and did not waste time between tools. The finding of cycles is that programmers focus on collecting new information, the cycles are shorter and the developer's invest a lot of time in each cycle. Several limitations to this pilot study need to be acknowledged. The collected data is limited because it does not take into account the other activities of the developers except for the activities that are performed with the computer and only a single case study is investigated in one development team [53].

I27. Maruyama et al. [32] underlie a new tool that visualizes source code called "CodeForest". The contribution of this tool is to focus on how the user will understand the program and not on the user's behaviour or the structure of the program. This tool combines various software metrics thus enhancing the user program comprehension. The user seeks source code by observing it is visual representations. In the first step, he should choose one option from

a different combination of 14 software metrics with six visual parameters of a forest tree to realize a specific representation, named "working set". After that, he interprets this visual representation then memorizes this. These three steps are done many times until he builds a mental model. The user can use annotations in order to help him in the memorization. "CodeForest" memorizes the written annotations and records of the executed actions of the user that may help him to understand the program. No study proves the effectiveness and efficiency of this tool [32].

I28. Jermann et al. [25] report a dual eye-tracking study to address the two following research questions: "1. is selection sharing beneficial for collaboration quality? 2. what happens during selections?". 40 pairs of students take part in this experiment. They are asked to execute two tasks of pair programming. In the first task, the rules of a game implemented in java should be described. Secondly, errors in the game implementation should be founded and fixed. During the study, the eye gaze, the speech ,and interface actions of each participant are recorded. Cross-recurrence is measured to detect the relationship between interaction quality and the degree of comprehension. Further analysis showed that gaze cross-recurrence is higher for good interaction quality, during spoken compared to silent episodes and during moments with selection compared to those without selection. Taken together, these results suggest that there is an association between gaze cross-recurrence and high interaction quality also between gaze cross-recurrence and episodes of referential selection. Another interesting finding is that the attention of the developer is attracted during shared and dual text selections [25].

4.5.4 Third iteration

I31. Chen et al. [27] investigate an approach to increase the awareness of the programmer to collaborate better and work with each other. An eclipse plug-in tool "Syde" is used to record all changes transparently done by the developer and to broadcast them in real time. Another tool "Scamp" is also employed to treat the data produced by Syde to support the developers to understand the modification executed in the system. Two multi-developer projects are developed in this study. "Wordcloud view" and "Bucket view" are used to enhance the awareness of the team. "Wordcloud view" is responsible for highlighting the classes of interest in the project and "Bucket

view" give an account to the developer who works on which part. The main advantages of using this tool are increasing the awareness, avoiding conflicts and duplicating work inside the project. Some significant limitations need to be considered. First, the small amount of the participants does not allow the author to have a solid conclusion. Second, the participants work in pairs which means there is only a simple path of communication. Third, they work together on the project. Fourth, the developers of this tool are also the people who assess the study [27].

I32. Meulen et al. [58] underlie a new technique to explore the visual behaviour of multiple users during a collaborative task around an interactive area. Four participants take part in this experiment. Each participant wears eye trackers. The goal is to represent their visual behaviour and to detect joint attention over different users during the study. This novel method permit to assess the visual behaviour of multiple participants in order to understand the collaborative behaviour profoundly on a multi-touch surface. However, the most severe disadvantage of this method is that it employs displays that transform the table as a landmark therefore a modification is needed [58].

I33. Chen et al. [9] present an evaluation of a visualization technique that enhances programming collaboration through mobile devices. This tool includes three views. It is dedicated to project managers and developers. Eye tracking is used to test the usability and effectiveness of this new tool. The eye gaze of the developer is compared with the focus of his eyes on the screen of the mobile device while executing a specific task. This tool is an enhancement of "team radar mobile". It uses a treemap visualization technique to have a hierarchical views, and it embodies two views one for the developer and one for the manager. The programmer view includes the team-mate names, the detailed performed programming tasks and the specific files of each team-mate. Real-time chat is also supported. Eleven participants take part in this study. Their performance (correctness and completion time) are measured during the achievement of the tasks. The results of the experiment show that the effectivity of this tool to display a big number of awareness information on the screen of the mobile device is achieved. However, the users are lost between the three views of the tool. In the developer view, they cannot understand which part is relevant and could not compare the information. Therefore, the improvement of this tool

is needed [9].

4.5.5 Fourth iteration

I41. Sharma et al. [50] report the interaction of the participants in a pair program comprehension task. Eye-tracking is used across different time scales to record the eye-movements of the developers. It was decided to distinguish the interaction of episodes into four layers. In this study, the author investigates the relationship between several layers at various time ranges. This research seeks to address the following questions: "1. how does the level of understanding relate the prevalence of different gaze episodes? 2. how do the types of gaze episodes relate to the types of dialogue episodes? 3. how do different dialogue episodes relate to different gaze transitions?". The procedures of this study are as follows: a pair of participants should give an account of the rules of a game implemented in Java. In this study, two types of understanding are distinguished: high and low level of knowledge. The purpose of the current study was to determine gaze and dialogue indicators at various time ranges in pair program comprehension task and to report the relationship between gaze and group cognition. The most interesting finding was the direct relationship between gaze and dialogue indicators at a different time. This finding has important implications for understanding the knowledge that inhibits program comprehension, besides, the collaboration that controls pair programming [50].

Chapter 5

Results

Chapter 4 describes the procedures and methods used in this review. In addition, an analysis of the retrieved papers is done. This analysis includes the author, the year ,the type of study, the general purpose, the research question, the approaches that are used and finally the results of each paper. This chapter will answer the research question of this review by synthesising, classifying and evaluating the identified data. Table 5.1 illustrates a summary of the main findings of all papers included in the study, with encompassing information about the paper, type of participant, task type and method to evaluate the study which has arisen in these selected papers, are provided in the next table. In the appendix, the complete summarisation of the literature review papers is available with author, year, environment in the table B.1. Task type is divided into three types namely program comprehension, debugging, collaborative programming.

Table 5.1: Summarisation of the literature review papers

Paper	**Type of participants**	**Task type**	**Methods to evaluate the study**
[44]	Professionals	Collaborative programming	Eye tracking + verbal protocol
[13]	Professionals	Collaborative programming	CAISE based CSE tools
[57]	Professionals + students	Program comprehension	Observation (think aloud method)
[10]	Students	Program comprehension	Video + sound are recorded + written notes are analyzed
[64]	Professionals	Debugging	Observation+ interview

Paper	Type of participants	Task type	Methods to evaluate the study
[5]	Professionals	Collaborative programming	Survey + interview + pre and post observation
[14]	Professionals	Collaborative programming	Eye tracking + video analysis + interview
[45]	Professionals	Program comprehension	Observation (think aloud method) + interview
[3]	Professionals + students	Debugging	Profiling tool + Observation (think aloud method) + interview + audio record + video of the screen + log of various navigations
[42]	Professionals	Collaborative programming	Eye tracking + screen capture with cursor of the test programs + facial videos of users
[29]	Professionals	Program comprehension	Survey + observation + interview
[62]	Professionals + students	Collaborative programming	Video recording + chat logs + survey + CVS repository
[43]	Professionals	Collaborative programming	Eye Tracking
[16]	Professionals	Program comprehension	Interview + observation
[40]	Students	Collaborative programming	IDE interactions
[6]	Students	Collaborative programming	Brain activity of the user
[54]	Professionals	Debugging	Eye tracking + video recording + observation (think aloud method) + audio record
[8]	Students	Collaborative programming	Syde
[17]	Professionals	Program comprehension	Observation (think aloud method) + log of code information
[53]	Professionals	Collaborative programming	PROM
[32]	N\A	Program comprehension	Notes + recorded actions
[27]	Professionals + students	Program comprehension	Average correctness + completion time
[58]	Students	Collaborative programming	Eye tracking

Paper	Type of participants	Task type	Methods to evaluate the study
[9]	Students	Program comprehension	Eye tracking + video recording + interview
[50]	Students	Program comprehension	Eye tracking + observation
[25]	Students	Collaborative programming	Eye tracking + video recording + logs of selection + audio record

Table 5.1 is quite revealing in several ways. There are various methods to evaluate the studies. Strong evidence exists that eye-tracking method has been used for a long time. They was used in studies from 2004 and even in 2017 studies. However, a new method that uses the brain activity of the user is deployed in 2018. The number of papers that are related to collaborative programming tasks are approximate to program comprehension tasks, but the number of studies that are related to debugging tasks is lower. Debugging tasks do not get the same attention among researchers due to the low number of publications and experiments on these topics. To get a better overview of the important information in the table, a pie chart is used.

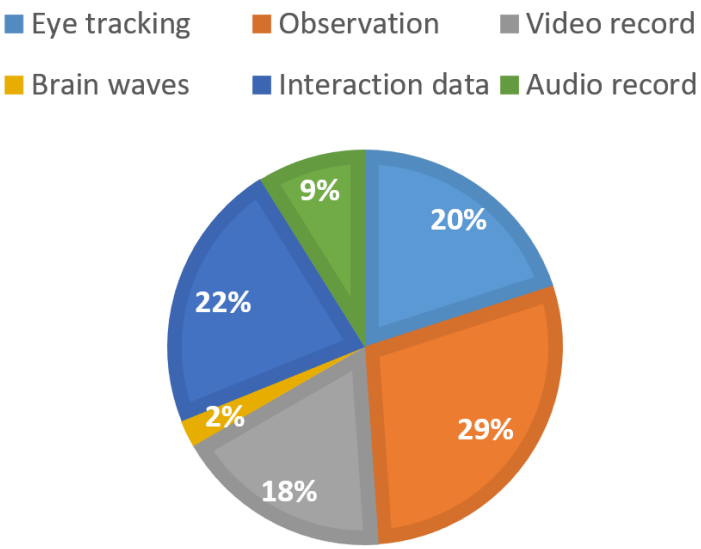

Figure 5.1: Evaluation methods

Figure 5.1 illustrates the employed methods in the studies with their

respective percentage. It compares most evaluation methods that are used in the studies. It is apparent from this pie chart that the most applied methods are observation and eye tracking to evaluate the attention and the comprehension of the user. Only 2% of the studies uses brain waves to evaluate the attention of the user because brain waves measurement is an innovative technology ; it appears for only a few years.

5.1 RQ1: Approaches to steer the attention of the user

In this section the results of the first research question will be outlined. Table 5.2 inspired by [62] summarizes the approaches that are used to orient user's attention in order to support him to have an overview on the whole project or to understand a part of a document. The rows of this table embody the tool or model, type of the tool, awareness source, awareness visualization, awareness filter. In the appendix, the complete table C.1 is available with additionally rows such integrated communication functionality, if it is available, and type of information.

Table 5.2: Types of awareness information inspired by [62]

Tool/ Model	**Type**	**Awarness source**	**Awarness visualization**	**Awarness filter**
Team-WATCH **[62]**	Standalone	Version control repository, local workspace and issue tracking system	3D visualization for each developer Standard view for all team member	Awareness information are chosen by the developer
Team radar mobile **[8, 9]**	Standalone	Local workspace	2D visualization for each developer and team leader	Awareness information are chosen by the developer
CRI model **[40]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	Awareness information are chosen by the developer
FAST-Dash **[5]**	Standalone	Version control repository, local workspace	2D standard view for all team member	N\A

Tool/ Model	**Type**	**Awarness source**	**Awarness visualization**	**Awarness filter**
Team tracks **[16]**	Standalone	Local workspace	2D standard view for all team member	N\A
Code-Forest **[32]**	Standalone	CodeForest automatically records a user's actions	3D visualization for each developer	Awareness information are chosen by the developer
Recording concerns using annotations **[57]**	Standard IDE of java language	Annotation are written by the user	Annotations are included in the code	Developer searches for specifics annotation with search function
Scamp and syde **[27]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	N\A
Pollinator **[64]**	Eclipse plug-in	Version control repository and local workspace	2D standard view for all team member	N\A
CAISE based CSE tool **[13, 32]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	Awareness information are chosen by the developer
Wear based filtering **[17]**	Eclipse plug-in	Version control repository and local workspace	2D standard view for all team member	N\A
Collaborative programming **[53, 42, 6] [3, 64, 10] [14]**	N\A	Direct communication	Shared visual attention or shared display	N\A
Eye tracking **[44, 14] [54, 43]**	Eclipse plug-in	Eye Tracker shared eye gaze	2D standard view for all team member	N\A
Text selection **[25]**	Eclipse plug-in	Selection of a part of the code	2D standard view for all team member	N\A

As shown in table 5.2, the approaches that are used can be standalone or plug-in. Besides, the awareness information can be extracted from various sources such as local workspace, Version control repository and/or recorded user actions, etc. To support the awareness of the developer, the art of visualization is also different. The visualization can be in 2D or 3D art. It can also be customized according to the preferences of the user in some tools.

Moreover, in some approaches, the user can select the type of information in which they are interested. In the next step, these approaches will be classified in order to get an overview of the similarities and gaps of each category.

The categorisation is based on the way of presenting the attention data to facilitate the program comprehension in the team. It can be shared through the same programming tool with the use of extensions, collaborative visualization tools or face to face collaboration. In the following subsections, each category will be explained in detail, highlighting the advantages and limitations.

5.1.1 Attention sharing by collaborative visualization tools

Card et al. [7] define visualization as "the use of computer supported, interactive, visual representations of data to amplify cognition", that means tools that help the users to understand a data through converting and assigning the data in visual context. As the proverb says, a picture is worth a thousand words. Visualization tools transform the data into information graphics. The collaborative visualization tools are a combination of visualization tools and collaborative tools. They are used to highlight the structure, and evolution of a program and display it to the team members [5, 62, 16]. Among the offered features, some visualization tools assist the user in combining software metrics and mapping them on visual parameters and allows the user to leave notes for recording the present comprehension [32]. Visualization tools can be divided into three kinds: desktop visualization tools, mobile visualization tools, and interactive surface visualization tools.

Desktop collaborative visualization tools

Desktop visualization tools, as the name suggests, conducts the visualization through a desktop. The principal advantages of desktop visualization tools are conflict detection, minimized cognitive load, simple coordination, and the maximization of team activity awareness. Moreover, the developer can check who opened a file, which file is being viewed and which modification have been done in this file, the verbal communication about the project is increased, a complete overview of the workspace is highlighted. Also, he can

extract and visualize only the relevant activity information to avoid mental overload. In addition, novice programmers can easily understand the shared information in the workspace [13, 5, 16, 17, 32].

Despite its efficiency and reliability, desktop collaborative visualization tools that share source code suffer from several significant drawbacks. For instance a high cognitive and mental workload are required for example in the case of CodeForest [32], where the user should memorize visual representation of the source code until he builds a mental model. Also, it is predestining for programmers that work in the same room and at the same time. Therefore only a small number of developers can profit from this tool. Consequently, it is not suited for big projects or remote collaboration [5].

Mobile collaborative visualization tools

Mobile visualization tool refers to the use of mobile devices such as smart phone or tablet to visualize the information related to the project. This method provides additional functions compared to the over kinds of visualization tools, such that the continuous awareness of the workers is assured. The developer can see i.e. what the file of interest is, the workload, who has modified or checked a file. This solution furthers the other visualization tools by supporting multiple views suitable for software developers as well as for the team leader. These tools are dedicated for program comprehension tasks, so that the correctness and a better completion time in the tasks are insured. Besides, mobile collaborative visualization tools hold ad-hoc collaboration [8].

One of the significant drawbacks of adopting this system is information visualization. It is challenging because all the relevant knowledge should be available for the user on the small display of his mobile device. Using treemap can be a solution to display the hierarchical information on the small screen. The coordination between sizeable collaboration is a difficult task. Also, it is challenging to maintain the balance between maximizing the performance and minimizing the power consumption of the mobile devices. The scalability and the readability are influenced by the flooded information [8, 9].

Interactive surface collaborative visualization tools

Sharing an interactive surface is an innovative approach to enhance collaboration. It consists of sharing a large-scale display; this can be a multi-touch tabletop [58] or a normal display [5]. All the team members can simultaneously use it to execute a specific task. This method represents a valuable alternative to sharing the same computer display in collaboration. The interactive surface allows a higher number of participants around the screen compared to the use of a single display. Therefore the joint attention of the worker is improved [58]. One advantage of the interactive surface is the accurate overview of the critical information, and the co-location of all the team members facilitate the efficient communication that is related to the project. Another advantage is avoiding disturbing events such as going to the other room or worse pointing system that delay the work.

However, Van der Meulen et al. [58] identify major drawbacks in this method. Using a multi-touch tabletop makes the table as landmarks. Another disadvantage is that the number of participants still limited. Moreover, the large shared display demonstrates the information of one person at a time. The majority of the developers regard this way of sharing information is monotonous and embarrassing[5].

5.1.2 Attention sharing by IDE plug-ins

IDE offers a high number of plug-ins that support collaborative programming and software visualization interface. Plug-ins in IDE can have different functions such as transforming an unfamiliar code or data interaction of the user to plots, trees, and graphs in order to help the developer to comprehend the program. Another essential function is the inclusion of user annotations or notes in the IDE to build a mental model, besides sharing the eye gaze or the text selection of the partner.

The aim is assisting the user in focusing their attention on the important part of the code and therefore in enhancing the collaboration. This approach is non-intrusive and lightweight. The key advantage of this approach is to expand the visual focus of the developer through the displayed information that are related to the modified files and the people who changed these files and exhibit statistics about the important part of the projects [27].

The main weakness of this approach is that the reserved space for

displaying the relevant informations is not suitable for exploring the data. The IDE interface includes a lot of heterogeneous information (classes, methods, visualization data etc.) thus the developer sometimes loses the overview. Besides, acquiring knowledge about the use of the IDE is necessary. Nevertheless, programmers need the manuals to solve some specific tasks [64].

As reported in the chapter 4, annotations or visualization features, eye tracking, text selection and wear based filtering can be used to enhance collaborative programming. Therefore, IDE plug-ins that support the user attention sharing are divided into five main features as follow: attention sharing by eye-tracking, attention sharing by text selection, attention sharing by annotations, attention sharing by interaction, and attention sharing by wear-based filtering.

Attention sharing by eye tracking

Eye tracking is a method to increase the awareness in collaborative programming through showing a persons visual focus of attention. It can be used in remote or real-time collaboration. The shared eye gaze supports the developer to understand, modify or solve a specific task. The eye gaze can be shared in different ways: by sharing the followed cognitive path of the eye movement of the developer [54] or by showing the part, where the partner is looking, and changing the colour when they are looking at the same position. One possible visualization is gaze plots. Another possible visualization of the eye gaze is heatmap. Figure 5.2, on the left side, displays an example of gaze plot. It consists of displaying the location, and the order of the focus of the user on a specific part in the code. Figure 5.3, on the right side, illustrates an example of heatmap. It reposes on showing how the visual attention focus is dispersed on the code or document.

D'Angelo et al. [14] identify several advantages of eye tracking. First, it increases the performance of a team in a visio-spatial domain by merely seeing and following the eye gaze of one worker such that in debugging. Due to eye tracking, a novice user can improve his skills by learning the way of program understanding and developing of experts. The second advantage of using this method is reducing cognitive workload because pointing to a specific part of the code become faster and more efficient so that the workers spend more time looking at the same place. Last but not least, eye tracking allows flexibility of work location by the possibility to coordinate the work

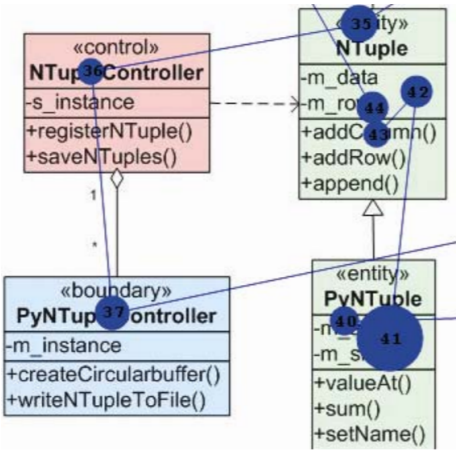

Figure 5.2: Gaze plot example according to Yusuf et al. (p. 2,[63])

Figure 5.3: Heatmap example according to Yusuf et al. (p. 7,[63])

in remote collaboration [14].

Eye tracking presents also disadvantages. One of these is that eye gaze is confusing if it does not follow a simple path [54]. The purpose of eye tracking can be challenging in some situations. The accuracy of the eye tracker decreases every time the user changes or moves around. That is why the calibration should be done in some cases [42]. Furthermore, the eye of the developer should be visible so that people who wear glasses, should wear a specific quality of glass or lenses due to their potentially disturbing reflections of the standard glasses or lenses so that the eye gaze will not be accurate. Also, eye tracker cannot reach all the collaborations. Sometimes there are outside the reach [9].

Also, in remote collaboration, additional tools should be used to achieve a correct synchronization [44]. Additionally the scalability of eye tracking is limited by the number of events in the display and the number of participants. A high number of fixations in the small time of interval and visual focus attention of multiple users makes the understanding of the program ambiguous.

Attention sharing by text selection

There are various methods to enhance the collaboration inside a team. One of these methods is sharing text selection to the partner by selecting a part

of interest in the code. So that the collaboration becomes faster and more efficient, the team will spend more time on effective collaboration. Dual selection can also be made to compare two parts of the code by the team or to confirm their partner reference by selecting the same region [25]. It is a kind of synchronous communication in the team.

One of the main advantages is to keep track of the attention focus of the partner and to highlight the relevant part of a document. The communication inside the team becomes very simple and precise through the selection because it is under less intentional control in comparison with eye tracking. The main limitation of sharing text selection is that the number of the developer in the team is not scalable. This approach is recommended to a small team.

Attention sharing by annotations

Source code annotations are tags used to decorate source code with meta-data. It supports the developer to understand the code rapidly by obtaining a mental model that encompasses helpful information about the code. The main advantage of annotations is that it has a positive effect on the comprehension, modification and the debugging of a program through the fast searching of the annotations. Also, it supports the developer to confirm hypotheses about the code, and to locate the features [37].

Thanks to annotations, the need for scrolling is reduced, and the orientation inside an unknown project becomes faster. The spot of annotations is easy and it can be an alternative of to-do comments. Another significant advantage is that the number of the developer that uses annotations is scalable and it can allow flexibility in the work. It assures an asynchronous communication between the team; thus the collaboration can be remote. Besides, program comprehension, correctness, and maintenance time are enhanced when annotations are used. The confidence of the tasks is not influenced when annotations are employed [57].

However, this approach has some limitations. One major drawback of this approach is that the annotation does not scale. There is an issue with the granularity; every line of code cannot be linked with their appropriate concern. A high cognitive and mental workload are needed in order to understand the concern because of mixing the annotations with the source code. Another issue with annotations is that it fails to be up to date [57].

Attention sharing by interaction data

IDE plug-ins that support attention sharing by interaction data, helps the user to comprehend the program by transforming this data into graphics like city metaphor, wordcloud, bucket view, etc.. The visualization of this data offers many advantages for example accelerating the program comprehension, getting a complete overview of the artefacts [40, 27]. Besides, the attention of the developer is guided on the relevant activities, which are highlighted. Who is changing, what is changing and when is changing in real-time are available to enhance the awareness of the developer [27]. These conveniences concern not only the developer but also the whole team so that the team members can be distributed. Plug-ins that share interaction data are non-intrusive and lightweight and support emerging conflicts. Avoiding redundancy and guiding the attention of the developer on the relevant parts are also supported [40, 27]. Task completion time is also improved [13]

The main downside of this tool is that the developer forget sometimes to connect the tool with the IDE [40]. This approach can be an alternative to face to face communication, especially for distributed teams [27].

Attention sharing by wear-based filtering

Wear-based filtering is a tool that encompasses computation wear and collaborative filtering. The term "computation wear" is used by McCandless et al. [23] to refer to recording user's interaction history and to broadcast them to the other members of the team. In broad terms, collaborative filtering is defined as a way of permitting the worker to rate or annotate the shared information inside the team. The purpose of this is to support the team-mate to search for easily relevant information [20]. Wear-based filtering contains three different visualizations to assure three different essential functions. The key advantage of wear-based filtering is supporting the user to inspect a definition without losing the current focus through a FAN list. Besides, this list can be ordered in the way that the most visited parts in the code display first. Furthermore, this approach highlights hot spots to guide the attention of the user on the parts of the code that takes the most attention by the other programmers. Moreover, an overview of the system components and their relationships is maintained through an automatic generated UML diagram [17].

One of the significant drawbacks to adopting this approach is that failure during programming effects the interaction data so that the accuracy of the visualization data will decrease. For example when a programmer often fixes a bug this will provoke interaction data that accentuate this part, and it will acquire more importance than it needs. Similarly, when the user falsely clicked on a part of the code, this part will appear as important for the other team members, but in reality, it does not do [17]. Another limitation is that the user needs a high mental and cognitive overload to adopt this tool.

5.1.3 Attention sharing by face to face collaboration

The joint attention in direct communication approach is very high. The direct communication is necessarily done in the same room and at the same time. Finger point and gesture are used in spoken dialogues to coordinate spatial attention [44]. The primary advantage is that direct communication eliminates disturbing activities [53]. The team is focused on the productive tasks; thus the task engagement will be increased. On the other hand, the mental workload will be decreased, and the programmer will work in a relaxed mode. Their level of concentration is higher than in solo work due to the moral obligation to not waste time in team working . Also, the developer tends to focus on new and innovative information instead of spending the time on understanding old information [6]. Task completion time is also enhanced [13].

One downside regarding this methodology is that some crucial information gets lost when one member of the team leaves the work. Another disadvantage is that the frequent interruption of the developer while improving their tasks will decrease their productivity. This approach is not recommended for big projects or teams because the worker will lose the overview and the communication will be impractical [53]. Besides, the capacity of the human to memorize all details is limited, and the transfer of knowledge and expertise is most of the time ambiguous and not evident [60]. Furthermore, in the case of pair programming as outlined in chapter 2, two different roles can be affected by each member, navigator and driver. The navigator spends a lot of time doing anything when the task is easy, or the driver has sufficient expertise and knowledge to solve it alone. Some solutions are outlined in the paper [43] for addressing this disadvantage . First, the collaboration should be partial. That means the pair should collaborate only

on the relevant parts. The navigator should spend his time reviewing the code. In this way, a continuous review will also be assured. This solution is only efficient when the pair does not share the same workspace [43].

5.2 RQ2: Attention measurement during collaborative programming

The outcome of the second research question will be highlighted in this section. Before that, the word "shared attention" will be defined. Shared attention is the fact of keeping a group of people focused on essential activities in a different domain. It can be enhanced by an external stimulus or an internal cognitive method [53].

In this section, attention data and attention measurement approaches in software engineering will be presented. These measurements are beneficial in enhancing methods that use these metrics to guide the attention or to assess other approaches. Various techniques are handled such as eye tracking, EEG, measuring the task/time completion and the error rate or observing the participants through a think-aloud method, audio, and video recording or analyzing the logs. Each measurement method can evaluate only specific attention data; that is why their classification will be based on the evaluation method.

5.2.1 Eye tracking

Eye tracking has become very popular in many research domains such as psychology, driving, reading, software engineering, etc. The fixations and saccades are the standard measurements to evaluate the visual effort of the user. Fixation refers to counting the number of times that the user is looking on Area Of Interest (AOI) and saccades mean the rapid movements of the eyes between the gaze of points [46].

Attention metrics with eye tracking [46, 24, 31] :

The estimation of the cognitive activity in software engineering using eye tracking is based on numerous metrics such as:

- **Fixation**: durations, Time To First Fixation (TTFF), total fixation time

- **Saccade**: velocities, amplitudes
- **Scan path**: distribution of gaze points
- **Visual effort**: pupil dilatation, blink number

The metrics of eye tracking are divided in this review into four main categories such as fixation, saccade, scan path, and visual effort. In the first category, the duration of the gaze point is measured to estimate the attention of the user for example high fixation means high concentration. Time to the first fixation means the period until the user detects the AOI and total fixation time indicates the duration of the focusing of the user on the AOI [9]. The second category is saccade, amplitude and the velocities are measured. Amplitude is "the angular distance the eye travels during the movement" [2]. In the third category, scan path refers to the distribution of the gaze points in the display.

Last but not least, the fourth category indicates the visual effort of the user over the pupil dilatation and the blinking number. Eye-tracking should be simple to use, to understand and non-intrusive [42]. The use of eye tracking as a measure in cooperation tasks is very challenging due to the obligation to check all the differences and the similarities between the developers and compare them. Fixations times and the number of fixations are generally used to a single feature of visual attention [42].

5.2.2 Multichannel EEG device

Multichannel EEG wearable device is a newly approach in software engineering to measure the attention of the user. This method is based on placing electrodes on the head in a precise position to record the brain waves of the human. This approach is different from fMRI. Multichannel EEG device assesses the brain performance during the software development in their daily work toward fMRI, and the developer should be in a lying position and see the code in a small mirror [6, 52].

Attention metrics with Multichannel EEG device [6]:

The assessment of the brain activity is typically evaluated through three different waves i.e. alpha waves, delta waves and theta waves [4].

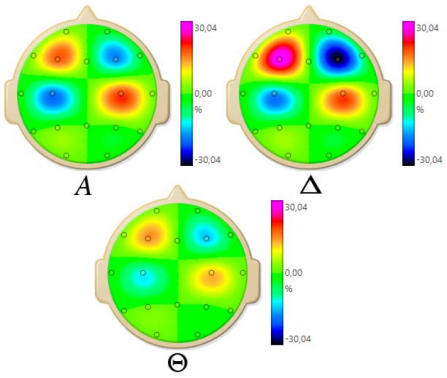

Figure 5.4: Topographic brain maps according to Busechian et al. (p. 3,[6])

Figure 5.4 illustrates a topographic brain maps for the alpha, delta, theta waves during pair programming. The brain waves are divided into different categories based on their speed [6]. In this review, only three types of waves are considered to measure the program comprehension of the developer. Alpha waves are responsible for the relaxation period [6]. Furthermore, delta waves give an account of the need of concentration to the execution of a specific task. Moreover, theta waves report the existence of distraction elements and the wish to eliminate it. The difference between brain activities when performing different activities can be obviously proved via wearable multichannel EEG devices [6].

5.2.3 Observation

Observation is as equally important as the other techniques for measuring the attention of the developer. It is considered as a data collection method. Audio/ video recording, logs and the reactions of the developers can be observed and analyzed to assess the attention during the collaborative programming. There is different metrics to evaluate the program understanding over observation such as correctness, completion time, joint attention, response to question about program comprehension, and number of switch between applications [58, 10, 17]. Joint attention means the focusing of the team member on the same task or on the same position in the screen. Additionally number of switches between the applications refers to the alternation of user between the IDE and the web to resolve problems [58, 10, 17].

5.3 RQ3: Correlation between collaboration and attention data

This part of the thesis responds to the third research question about the correlation between collaboration and attention data. After analyzing the papers, a positive correlation between cooperation and attention data has emerged. The measurement of attention data can be measured through different tools as explained in the previous section.

Pietinen et al. [44] highlight that the efficiency of the collaboration can be examined over the number of overlapping fixations. Problems in comprehension can cause higher joint fixations, long gaze duration, and high mean fixation. Contrarily, high rate of overlapping fixations can be the effect of a powerful collaboration [44]. Also, Sharma et al. [50] point out that the degree of comprehension influences the duration of the joint visual focus of the gaze of the developers. They argued that a high duration of joint attention implicates a high degree of understanding in the team. Besides, the joint visual focus influences the collaboration so that persons with a high level of understanding devote more time to collaboration. In the opposite direction, the observation is also right, a low level of understanding refers to lower time focusing together . The correlation between collaboration and the attention is also exemplified in work undertaken by Busechian et al. [6]. They prove that working in a team improves the focus on a specific task by eliminating disturbing activities. Moreover, they confirm that the brain waves of pair programmers are different from the brain waves of solo programmers. As mentioned in the previous section, brain waves is a novel approach to measure attention. Sillitti et al. [53] also confirm that pair programming has an impact on the attention of the developer. They approve that the focus of the developer on their tasks is enhanced, such that the time to finish a task is decreased, their productivity is increased and switching between tools is also decreased [53].

The evidence from these studies confirms that attention guidance helps the developers to find and navigate relevant code on the project quickly, therefore the cooperation is optimized. Similarly, cooperation increases the attention orientation on important tasks.

5.4 Threats to validity

The purpose of this literature review is to summarize and to evaluate the retrieved approach that are related to steering the attention of the developer during collaborative programming. Nonetheless, there are threats to the validity of the results and findings of this review. These threats are divided in four categories according to Wohlin et al. [12] i.e. internal, external, construct and conclusion validity.

Internal validity:

A deviation from the guidelines of Kitchenham [26] is done because the execution of the research is made by only one person and this search is done manually. Another threat is inclusion and exclusion criteria. They are defined to avoid subjective decisions, but the choice to include a paper to the review is still subjective. Moreover, the classification is based on the way of presenting the attention data to facilitate the program comprehension in the team. The choice of this classification is subjective. This classification may affect the interpretation of the results.

External validity:

The fullness of the studies depends on the used keywords but there is a limitation in the number of characters of the search strings that should be inserted in the databases. Otherwise, there is a limitation in the search engine and databases as the syntax and standards varies among databases. This can lead to miss some relevant papers.

Construct validity:

Correctly analyzing all articles is challenging because synonyms and ambiguous terminology are used. Due to these points, other researchers may decide differently. Also another search string that includes different synonyms would have different results. Five databases are selected, in order to increase the probability to reach the most relevant papers.

Chapter 6

Evaluation and discussion

Reviews perform an important role in synthesizing key research results and highlighting new research areas for researchers. This review proposes a classification and a description in chapter 5 of the awareness approaches that support steering the attention of the developer on the important parts in the program. In this chapter, a comparison between the approaches will be made.

This chapter is composed as follows: firstly, the evaluation feature will be defined. Secondly, approaches will be assessed and compared on the bases of the predefined criteria. And finally, a discussion will highlight the gaps and similarities with the help of different visualizations.

6.1 Evaluation features

The aim of collaborative visualization tool, IDE plugins that support attention sharing and face to face communication, is enhancing the awareness inside the team and improving the program comprehension. Therefore, the evaluation features are oriented to the awareness of human activities and program comprehension. Dourish and Bellotti [18] refer to awareness as "an understanding of the activities of others, which provides a context for your own activity". The evaluation criteria are based on the factors that influence the cooperation between the team and that facilitate the program understanding. As explained in the motivation in chapter 1, it is clear that the goal of this review is to summarize and assess all of the approaches that are related to steering the attention of the developer during

collaborative programming in order to get an overview of the process of software development and to enhance the program understanding. To this end, the features for the evaluation that are oriented to awareness of human activities are described in the classification of Gutwin et al. [21] and the classification of Storey et al. [56]. In this review, a combination of both classifications is done. The choice of the included features is based on the available information in the retrieved papers. For example, the cost feature, by Storey et al. [56] classification, cannot be included in the evaluation due to the lack of the costs in the majority of the retrieved papers.

The features defined by Gutwin et al. [21] are as follows: change detection, intention, activity, location, identity, and sphere of influence. Storey et al. [56] features are namely time, kinds of view, navigation view, interoperability, scalability, learning. Additionally to the feature of Gutwin et al. and Storey et al., the features that are related to steering the attention and enhancing program comprehension are the contribution of this review. These features are deduced from the gaps that are detected in chapter 4 by the analysis of the retrieved papers such as familiarization with the project [29, 40], conflict detection [62], and mental model [45, 57]. The rest of the features are inferred from the goals of orienting attention such as distraction elimination [6], mental workload [29], cognitive workload [57], and productivity (time/task accomplishment) [29].

Since different features are used for the comparisons, a categorization is meaningful. The features will be classified, based on six key dimensions such as time, kinds of view, navigation view, effectiveness, efficiency, and expressiveness. Time refers to the time of occurrence of the awareness activity. It can be past, past present or/and historical data [56]. Kinds of view means the types of information awareness illustration. It can be annotation, graph view, or statistical view [56]. Navigation view indicates the art of user's navigation through the visualized information. They can have a detailed or an overview of the whole visualization. The view can be also coupled or zoomable [56]. The effectiveness deals with the adequacy to accomplish the expected goals, this definition is given by Storey et al. [56]. Froehlich and Dourish [18] confirm that interoperability is important because awareness system should work in combination with other tools. Scalability is also essential in order to decide if the approach holds a large software project. Learning deal with the manner of adopting the approach. Interoperability,

scalability, and learning will be part of the second dimension, which is effectiveness [56]. Efficiency is defined as the effect of using a specific tool to support program understanding and to enhance collaborative programming. The efficiency is measured over productivity, mental/cognitive workload, and distraction elimination. Maletic et al. [30] define expressiveness in program comprehension as the specification about who, how, why, and what is attempting to understand a specific object. Person's interaction with the workspace is the centre of interest of this dimension. According to this definition, the following features will belong to expressiveness.

- **Change detection [21]:** the user can detect the modifications that are made in the code.
- **Conflict detection [62]:** the tool can help the user to detect conflict when two users or more, work on the same file.
- **Mental model [45, 57]:** the user can quickly build a mental model to have an overview.
- **Familiarization with the project [29, 40]:** the tool assists the newcomer user to have an overview of the unfamiliar code.
- **Intention [21]:** the user can know the goals of their team mate
- **Activity [21]:** includes information about the achieved activities of each member of the team.
- **Identity [21]:** incorporates information about the person who modified a specific file.
- **Location [21]:** consists of knowing the position where the partners are working.
- **Sphere of influence [21]:** refers to the positions where changes should be made.

6.2 Approaches evaluation

In the section 6.1, evaluation features are defined with the intention of evaluating the outlined approaches such as collaborative visualization tools,

IDE plug-ins and face to face collaboration from chapter 5 in this section. Each group will be evaluated in the table 6.1. The key dimensions of the evaluation are presented in the rows of table and divided in six parts. This are: time, kinds of view, navigation view, effectiveness, efficiency and expressiveness. Each part of the rows is divided into sub-parts. The column of the table encompasses the subcategories of each approach.

This approaches are made to support different kinds of tasks such as program comprehension, collaborative programming and steering the attention of the developer in relevant parts of a code or document. The comparison of these approaches is based on the type of visualization (kinds of view and navigation view), type of displayed information (present, recent past or historical information), the efficiency, the effectiveness and the expressiveness of the approaches to achieve the goals. The differences between collaborative visualization tool, IDE plug-ins and face to face collaboration are highlighted in table 6.1. It illustrates only the three classes of the approaches due to the lack of place in the page of the report. For more detail the complete table is available in the appendix A.1, where the sub-categories of each approach are compared. Three colours are used in the table as follows: red for 0% to 33% , yellow for 34% to 66% and green for 67% to 100%. The percentage means the availability of the feature in this approach. For example: only two approaches out of five approaches uses the present time in the IDE plug-ins. $\frac{2}{5}$=40% so the corresponding case of the table must be in the colour yellow. The assessment is based on the features that are included in each approach. For example the tools of desktop visualization are various and not all the tools offer the same features. So in this case, a feature is considered available, if it is present in one of the selected tools in this approach (best case is taken). In table 6.1, the collaborative visualization tools will be evaluated.

It is apparent from table 6.1 that most collaborative visualization tools support various time information, different kinds of view and navigation view. Graph view and statistical view are derived data. This data helps the developer to be aware immediately of all changes, the process of the work and particularities that should be known. This type of view allows the developer to answer the who, what and how questions. Navigation view aid to reduce the cognitive and mental workload. A complete overview of the project is assured over the zoomable view. Under these factors, the efficiency is

Table 6.1: Evaluation of collaborative visualization tools

Evaluation features		Collaborative visualization tool	IDE plug-ins	Face to face collaboration
Time [56]	Present			
	Recent past			
	Historical			
Kind of view [56]	Annotation			
	Graph view			
	Statistical view			
Navigation view [56]	Overview/detailed			
	Coupled view			
	Zoom-able view			
Effecti-veness [56]	Scalability			
	Interoperability			
	Learning			
Efficiency [29, 57, 6]	Productivity			
	Mental workload			
	Cognitive workload			
	Distraction elimination			
Expressi-veness [21, 29, 40] [45, 57, 62]	Change detection			
	Conflict detection			
	Mental model			
	Familiarisation with the project			
	Intention			
	activity			
	Identity			
	Location			
	Sphere of influence			

enhanced as well as that some features of the expressiveness are also assured for example, activity, identity, location, identity, the sphere of influence, change detection and familiarization with the project. Mental model and conflict detection are not supported because collaborative visualization tools are based on the interaction of data and changes in the local workspace of

each developer and not on the source code.

Furthermore from table 6.1, it is evident that IDE plug-ins which support awareness combine recent past and historical information except eye tracking and text selection that use real-time information. The coupled view is assured in this approach. Its main advantages are comparing two or more views in real time and avoiding switching between tools, but one of the main limits is the high cognitive workload that is needed to solve a task. These tools are not scalable, and learning is not always natural. The developer should acquire knowledge in order to profit from this approach. Change and conflict detection are seldom assured; three of nine features of the expressiveness are not covered. The causes of these results are that these tools do not use derived data in most cases. That is why a high cognitive and mental workload are needed because the user alone should build a mental model and interpret the displayed data.

Moreover, table 6.1 sets out that in face to face collaboration, real-time and recent past information are used to enhance collaborative programming on account of the direct communication between the collaborators. There is no view in this approach because it is consists of programming and communicating in the same room together. It is efficient and expressive but not scalable as explained in the chapter 5.

6.3 Gaps and similarities

In order to have a better overview on the similarities and gaps between the approaches, visualizations are used. The definitions of the features criterion are pointed out in the section 6.1.

6.3.1 Modified Venn diagram

A modified Venn diagram 6.1 is adapted to highlight the logical relation between the reviewed papers and the used approaches to steer the attention of the developer during collaborative programming. Each circle symbolizes each approach.The overlapping between the circles means that the authors include more than one approach in their paper according to the available classification in chapter 5. This can be seen in the case of D'Angelo et al. [14], which uses eye-tracking to enhance the communication between the pair. So in this case, the paper [14] should be included in the overlap of

eye-tracking and face to face collaboration, similarly for the other papers. Different colours are used to facilitate the visualization of each set. The reviewed papers are linked to simplify the access to the papers.

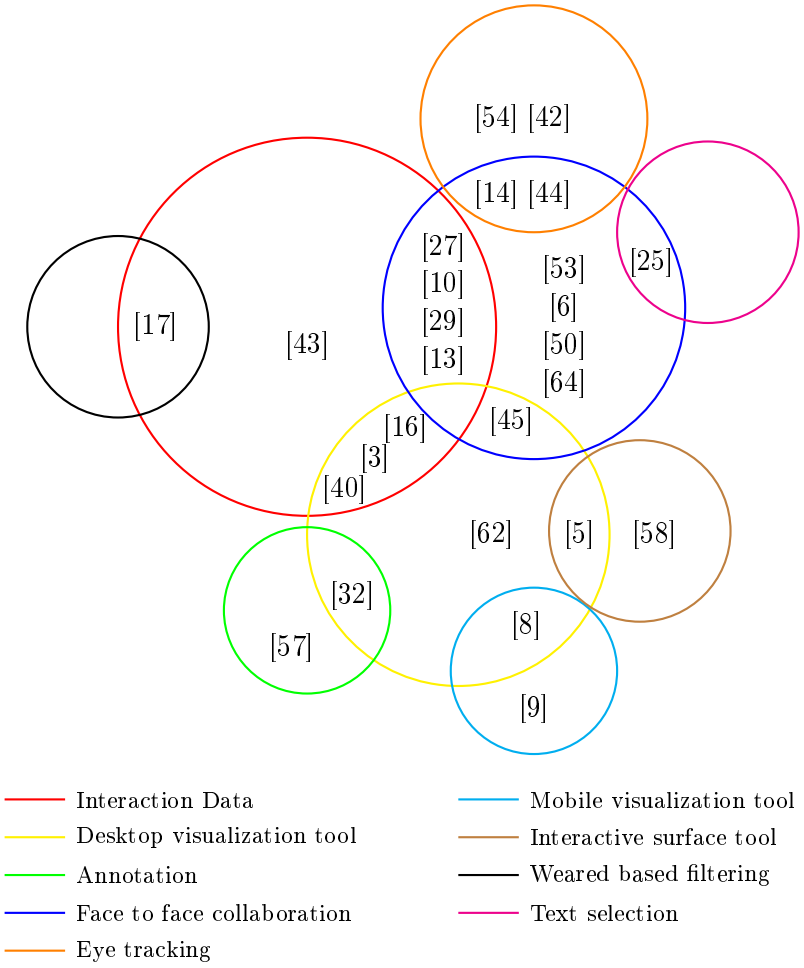

Figure 6.1: A visual summary of all the papers used in the literature review

What is interesting in the figure 6.1, is that the most of the papers are situated in the overlapping between interaction data and desktop visualization tool approaches and the overlapping between interaction data and face to face collaboration. This finding can be explained by the importance of interaction data to visualize important information. As explained in chapter 2, software visualization permits to display the code and the relation between the code. In desktop visualization tools, interaction data and source code are used for example to conclude statistics and display

graphs that help the programmer to orient their attention on crucial parts in the code. Additionally, the overlap between annotation, interaction data, and desktop visualization tools can be explained through the importance of combining annotation with other approaches to address the drawbacks of the annotations mentioned in the chapter 5. As known, annotations can sometimes be outdated and not sufficient to steer the attention on the relevant information. To solve this problem, interaction data and software visualization are applied to assure more efficiency, effectiveness, and expressiveness in real time. Also, the overlap between mobile visualization tool and desktop visualization tool can be justified over the prerequisite of having a continuous awareness when the developers or the team leader are out of office. The same version of desktop visualization tool is available in mobile visualization tool to increase awareness, similarly, with the overlap between the interactive surface visualization tool and desktop visualization tool. Interactive surface visualization tools are a complementary version of desktop visualization tool.

Furthermore, the overlap between wear-based filtering and interaction data can be demonstrated through the indicated definition of wear-based filtering in chapter 5 as follows recording user's interaction history and broadcast them to the other members of the team like in the paper of Jbara et al. [23]. From this definition, the strong correlation between wear-based filtering and interaction data can be explained. Last but not least, the papers that report eye-tracking and, similarly, text selection studies to steer the attention of the developer include not only eye tracking or text selection but also face to face collaboration. It can be demonstrated that eye tracking or text selection are not sufficient to explain and to maintain an overview of the software development process. Another approach should be combined in order to achieve the knowledge transfer, complete overview of the whole project and detailed view on the project process.

6.3.2 Radar chart

The goal of the visualization in this section is to display and compare all the nine approaches based on the defined features in the first section in one figure. Due to the multidimensionality of these data, it is very challenging to display it. Therefore, the choice of a radar chart. It is a less complicated visualization method in comparisons to the other visualization methods such

as parallel coordinates, and Chernoff faces. Radar chart is used to summarize all the approaches in the same chart. This chart is used to compare multiple quantitative variables. One advantage of these visualization is to identify similar values or outliers amongst each variable. Besides, it is used to display the performance of each variable. The variables are represented in each axis of the chart. In this case, six variables are available. This illustrates the feature evaluation of: time, kind of view, navigation view, efficiency, effectiveness, expressiveness. The definitions of these features are indicate in the first section of chapter 6. Each feature value is plotted along its individual axis. All the variables in a dataset are connected together to form a polygon. Each polygon represents one approach. In the chart 6.2, nine polygons are displayed. The radar chart encompasses three levels that are represented through the degrees 1, 2 and 3. The same principle as with the colours in the previous section. The percentage means the availability of the feature in this category. Three stages are used in the spider chart as follows "1" for 0% to 33% , "2" for 34% to 66% and "3" for 67% to 100%.

From the spider chart 6.2, the approaches can be compared. As can be identified from the figure 6.2, desktop visualization tool includes the most features with degree three. In contrast to annotation, it includes the lowest number of features with degree one. Besides, desktop visualization tool and mobile visualization tool have a lot of common segments, which means a lot of features in common such as efficiency, expressiveness, time and effectiveness. Besides, the polygons of eye tracking and face to face collaboration have five features in common such effectiveness, navigation view, kind of view, expressiveness, and efficiency. It is apparent from this chart that kind of navigation and kind of view are of little interest.

This chart is quite revealing in several ways. Firstly, unlike the other modified Venn diagram, the approaches can be directly compared according to their features. There was a significant resemblance between text selection and eye-tracking; both approaches have the same features with the same degree except for the feature time. By eye-tracking, the time can be present, recent past and historical, but by text selection, the time can be only present. In the same way, a desktop visualization tool and mobile visualization tool are identical for in most of their features except for kinds of view. This observation can be related through the small screen of mobile devices; it can not support many kinds of view.

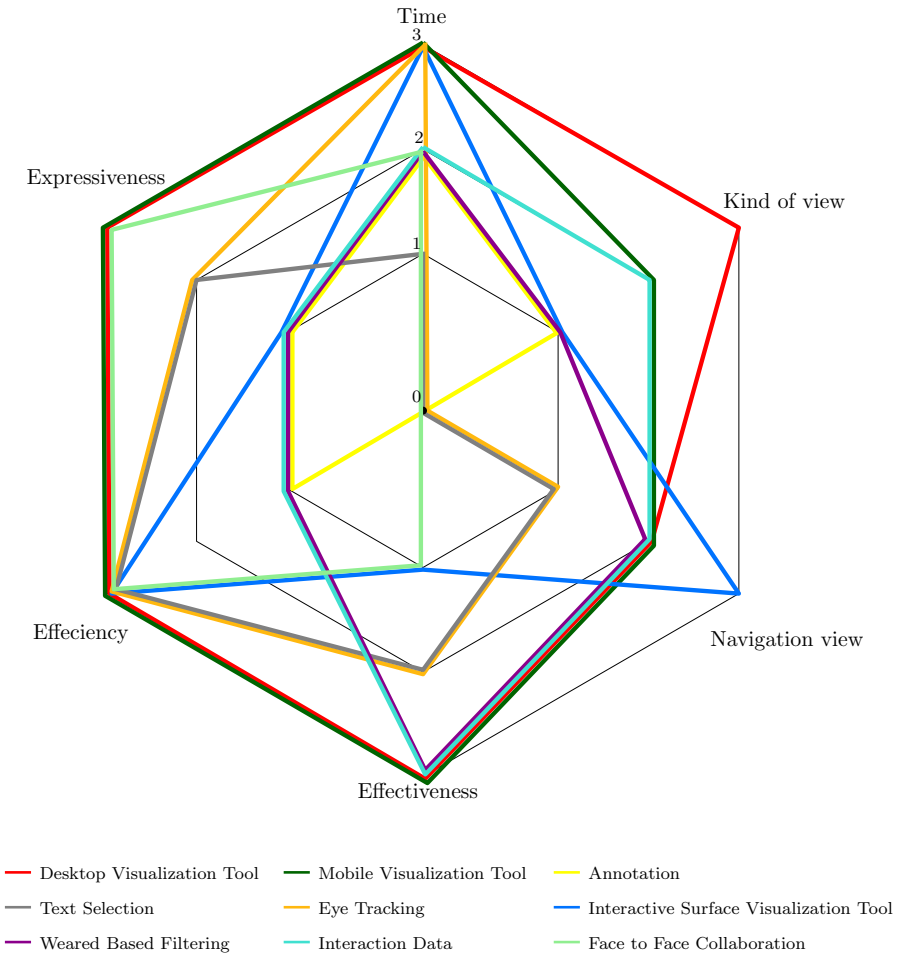

Figure 6.2: Spider chart for the assessment of the approaches

Interestingly, interaction data and wear-based filtering have the same number of features with the same degree except for kinds of view. This observation can be elucidated by choice of the researchers to include more features such as graph view in interaction data approach and not in wear-based filtering approach. Furthermore, it is apparent from the name that face to face collaboration does not support kinds of view and navigation view. For more details about the features that are included in each approach, table A.1 in the appendix illustrates the similarities and the differences.

6.3.3 Stacked bar chart

The comparisons between the approaches will be achieved with the support of a stacked bar chart. These comparisons will be performed between the subcategories of each evaluation criteria (effectiveness, efficiency and expressiveness). The choice of this visualization is based on its characteristics to show the parts of multiple components. Besides the simplicity of understanding it; the reader can better compare the approaches with each other. Three stacked bar charts are needed to assess the approaches according to the effectiveness, efficiency and expressiveness that are defined in the first section. The data is extracted from the table in the appendix A.1. The percentages in the following graphics are calculated through the sum of the available evaluation feature in all the categories divided by nine categories then multiplied by 100.

- **Effectiveness**

According to Storey et al. [56], effectiveness can be defined as the achieved goals with the help of a specific approach. As pointed out in the first section, three objectives are specified, such as scalability [56], interoperability [56], and learning [56]. Scalability refers to the ability to scale the project using this tool. Interoperability is a property that permits unlimited sharing of resources between various end-devices. And, learning refers to the rapidity of adopting a specific tool without difficulties. The stacked bar chart in figure 6.3 shows the components that assure the effectiveness in each approach.

From chart 6.3, it can be seen that desktop- and mobile visualization tools are the most effective. They assure the scalability, interoperability and easy learning. But annotation is the less effective because it is not scalable, not interoperable and difficult to learn. 66% of the approaches are easy to learn and interoperable. This observation can explain the overlap between annotations, interaction data and desktop visualization tool in the modified Venn diagram 6.1. To benefit from the advantages of annotations and to avoid its drawbacks, annotations is combined with other approaches.

- **Efficiency**

Efficiency can be defined as follows: the effect of using a specific tool to support program understanding and to enhance the collaborative

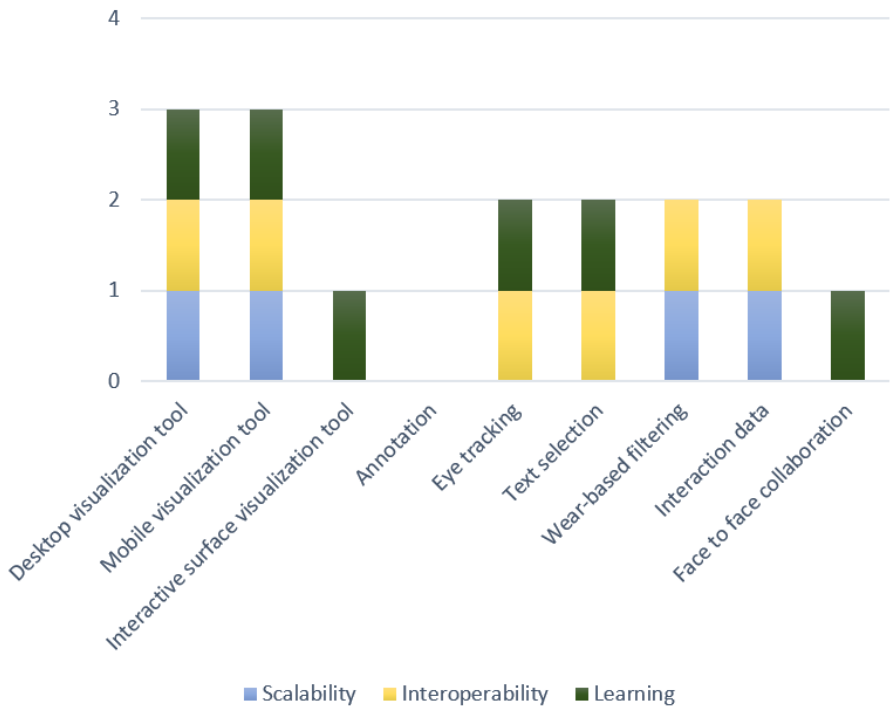

Figure 6.3: Effectiveness of the approaches

programming. The efficiency is measured over productivity [29], mental workload [29], cognitive workload [57] and distraction elimination [6].

The stacked bar chart 6.4 shows that 100% of the approaches help the user to eliminate distracting elements. These results assures and confirms that the first goal of this tool to orient the attention of the developer is achieved and the developers are concentrated on their work. 88% of the approaches make developers more productive in terms of completion time and error. 66% of the approaches does not need high mental and cognitive effort. Desktop visualisation tool, mobile visualisation tool, interactive surface visualisation tool, eye tracking, text selection and face to face collaboration provide the same advantages in terms of efficiency. The less efficient approach is annotation.

- **Expressiveness**

Maletic et al. [30] refer to expressiveness in program comprehension as the specification about who, how, why, and what is attempting to understand

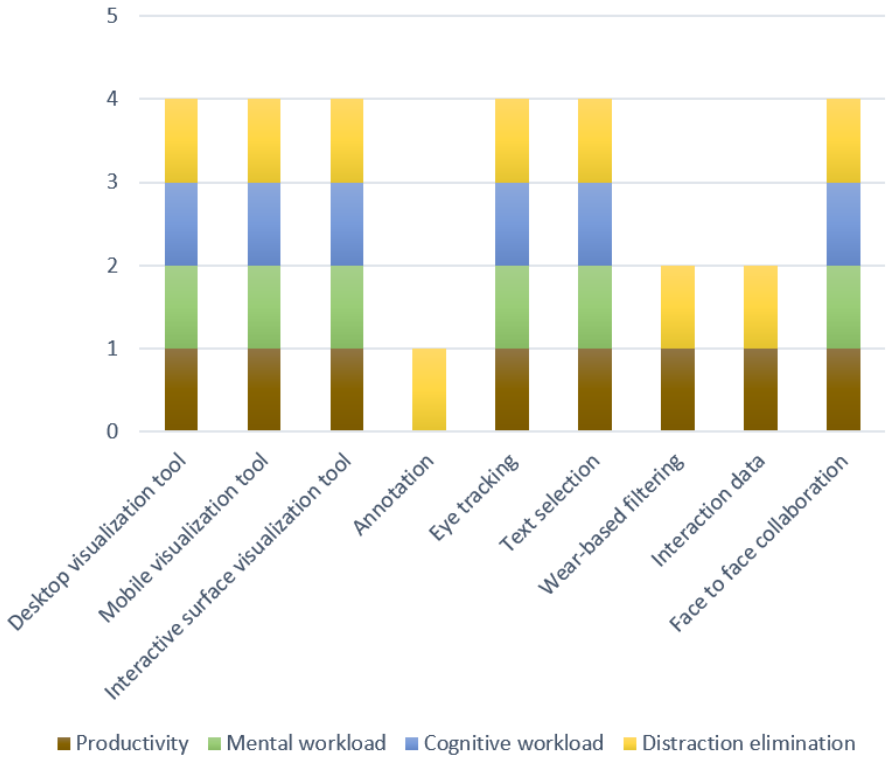

Figure 6.4: Efficiency of the approaches

a specific object. The interaction of the developer with the workspace is the major of interest of this dimension. The expressiveness of the approaches that share the attention of the developer is measured over nine features consisting of change detection [21], conflict detection [62], mental model [45, 57], familiarization with the project [29, 40], intention [21], activity [21], identity [21], location [21] and sphere of influence [21].

The stacked bar chart 6.5 displays that 88% of the approaches help the user to be familiar with the project. 77% of the approaches support the user to locate in which position in the code their team-mate are working and also 77% of the approaches give an account of the position where changes should be made. Only 22% of the approaches help the developer to detect the conflict and to know the intention of the others in the group. Interestingly, desktop visualization tool and face to face collaboration, eye tracking and text selection provide the same advantages related to the expressiveness.

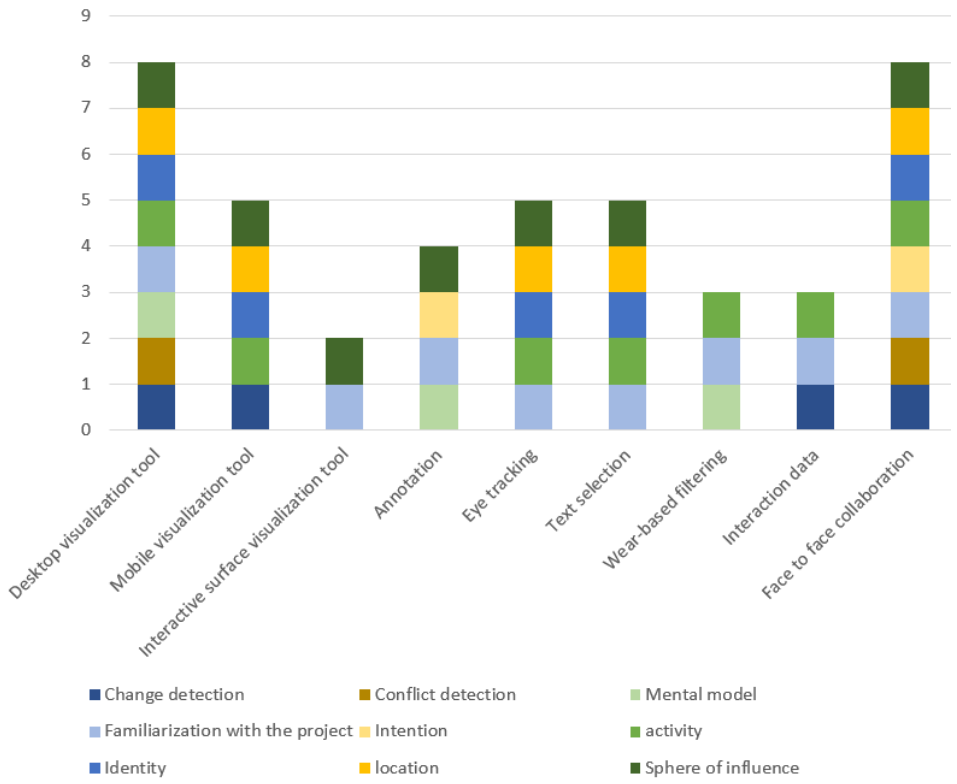

Figure 6.5: Expressiveness of the approaches

6.4 Discussion

This section aims to highlight the gaps in the retrieved works. The first gap in the retrieved citations is the graphical representation versus the textual representation in guiding the attention of the developer. The graphical representation is employed to facilitate the program understanding of the developer, but in some cases, this visualization complicates his work for many reasons such as: graphical representation is based on the interaction data or the source code. This data can be influenced by failure done by the developer like spending a lot of time trying to understand a part of a code. The interaction data will be respectively influenced, and this part will be represented as an essential part but in reality, it is only a comprehension problem, and it has no importance. Besides, some visualizations are not simple to understand such as Codeforest. The developer must memorize specific sets various times until he builds a mental model such in the paper

of Maruyama et al. [32]. The textual representation is outdated most of the time and is not scalable as with annotations. For these reasons, a mix of textual and graphical representations should be done. The modified Venn diagram 6.1 illustrates few cases of overlapping between textual and graphical approaches. The researchers are more concentrated on improving specific methods like eye-tracking, desktop visualization tool and not trying to combine the approaches except for the case of wear-based filtering.

The second gap is the considerable difference between research and practice. The majority of the studies are done in a controlled environment. They do not consider the needs of the developer, for example, the intention of the team and conflict detection. These tools orient the attention of the developer in real time. They do not consider that he can concentrate on more important parts and this will deviate his attention on the wrong way. Giving the developer a choice, to filter when he is ready to know relevant knowledge about the project or not, it can be a key to solve this limitation. Recording the data that steers his attention may be another solution for some cases like by eye-tracking, and text selection. The programmer alone should choose when and where he would like to use these approaches.

In some cases, these tools complicate the work of the developer instead of facilitating it in obvious tasks or when the approach is more complicated to understand then the program itself. The expertise and knowledge of the developer should be taken into account when such methods are developed. For example by interactive surface tools, the majority of the developers that take part in the study consider that the large screen is tedious and cumbersome to use, is given by Biehl et al. [5]. In the case of the eye-tracking approach, gaze plots and heatmap impede the workload of the developer due to displaying it in the background of the code. Besides, the used colours of these approaches may not be suitable for all the team for example, it would be unsuitable for people who have red/green weakness as pointed out by Ahrens et al. [1]. The success of the attention guidance depend on the type of task and programmers knowledge and expertise. Researchers should treat these dependencies with a lot of detail to provide approaches dedicated to each team.

Chapter 7

Summary and outlook

This final chapter draws upon the entire thesis, gives a brief summary and includes a discussion of the findings implication to future research into this area.

7.1 Summary

The aim of this thesis entitled "literature review and concept for cooperation in software development" is to perform a literature review based on the guidelines of a systematic literature review with snowballing research procedure. The results from this literature review provide the researchers a better understanding of the outlook of steering the attention of the developer on the important part in the code or the document during collaborative programming. The fetched approaches are categorized based on the way of sharing attention. The pursued categories are collaborative visualization tool, IDE plug-ins, and face to face collaboration.

Also, different techniques like eye tracking, multichannel EEG devices that are used to measure the attention of the developer during program comprehension, are summarized, and additionally the interplay between collaboration and the attention of the developer is explained.

Moreover, the retrieved categories are described and then compared according to the following criterion: time, expressiveness, effectiveness, kind of view, navigation view, and efficiency. For the comparisons, different visualizations such a modified Venn diagram, spider chart, and stacked bar charts are used to better demonstrate the gaps and the similarities between

the approaches.

Furthermore, discussion and interpretation are held. One of the main findings is that interaction data is essential to increase awareness in the team. Most of the time eye-tracking studies are combined with other approaches. This combination can be explained through the necessity of other tools to support the programmer to have an overview of the whole project, and the eye-movements of his team-mate is not sufficient to increase his awareness and to transfer his knowledge. Challenges and gaps are identified in the area that opens opportunities for future research. Additionally, this review can support practitioners and researchers in the identification of important challenges and the definition of lightening approaches that have already been tested in controlled or industrial settings spanning these 15 years.

7.2 Outlook

The aim of this review is to summarize and to assess the approaches that guide the attention of the developer, and then to find the gaps and similarities. In this section, recommendations for future studies on the attention guidance are outlined.

Future research should consider the potential effects of real-time attention transfer on the programmer's performance more carefully. For example, real-time information are not always suitable for the daily work and it can steer the attention of the developer in a negative way so that he can miss another critical task that would be made before switching to another task. One possible solution is to record this information in order to see it when he needs it and if possible several times to solve a bug or to enhance his comprehension. It is also recommended to more closely consider the intention of the team and conflict detection because they are essential to speed up the collaborative work.

In addition, a combination of the approaches should be done for example eye tracking and text selection. For example, the eye movement helps the developer to follow the path of the eye movement of his team-mate, but text selection would be helpful when a specific part in the code should be highlighted because, as known, eye tracking is sometimes ambiguous. Collaboration can arise outside the reach of eye tracker so that there is not shared attention shared to it [44]. So through the combination of multiple

tools, the drawbacks will be addressed.

Moreover, historical data are important to make statistics about the most parts that acquire the most attention. But the parts that have not gained much attention are forgotten, and this can cause problems. The tools should also advertise the team that a significant part of the code can be missed by mistake. Another limitation of the use of historical data is that the failure of the developers influences the interaction data [17] so that the attention of the developer will be steered in a false direction. Further studies should investigate in more detail how to filter the interaction data that will be transferred to drive the attention of the developer.

Standalone visualization tools should be automatically connected with the development environment because of the forgetting to connect it when starting developing, and this can influence the attention information [27].

Many programmers do not use tools that steer their attention to enhance collaboration and their performance because they do not know these tools. They are accustomed to develop without any support, or they think that it is difficult and not helpful enough [45]. The research should be more concentrate on the dependency of the success of the approach on type of the task, developer knowledge, developer expertise, and size of the team. Therefore, there is no one-size-fits-all strategy [29].

Consequently, future research should be conducted in more realistic settings to shed light on this gap between practice and research. Customized tools that are adapted to the real needs of the developer should exist. Also, the attention measurement should be more objective in deciding about the effectiveness of an approach. For these reasons, researchers should focus on the use of accurate measurement tools like EEG device multichannel, eye tracking, and fMRI rather than observation. The goodness of the attention measurement influences the virtue of the approaches that guide the attention. These measurement tools are employed not only to assess the efficiency of the tools but also to transfer the knowledge. Ahrens and al. [1] exemplify this in their work which is based on transforming the gaze data and the number of fixation into heatmap and class name colouring. This fixation number gives an account of the importance of the related part of the code [1]. The main problem of this study is that the code readability, the search for the relevant part in the code, and the cognitive load are impaired from the use of heatmap. Plug-ins can be integrated into the existing programming

language, which provides a different view, to address this problem. For example, a view can provide the developer awareness information like the list of the named classes with their ranking of importance. This view can be a better alternative to colouring the name of classes. In this way, the developer can be self-perpetuate and his attention will not be distracted. Complementary, another view should be used to visualize the important parts in the code through 3D representations, where the developer can zoom or scroll in it, to see more detail when he wants. In addition, the heatmap can be used in another view, when he needs to know the specific parts of interest. This view will address the program of the positioning of the heatmap in the background of the code.

Bibliography

[1] M. Ahrens, K. Schneider, and M. Busch. Attention in software maintenance: An eye tracking study. In *Proceeding of The 6th International Workshop on Eye Movements in Programming (EMIP)*, Montreal, Quebec, Canada, May 2019. IEEE / ACM.

[2] A. Bahill, M. R. Clark, and L. Stark. The main sequence, a tool for studying human eye movements. *Mathematical Biosciences*, 24(3):191 – 204, 1975.

[3] S. Baltes, O. Moseler, F. Beck, and S. Diehl. Navigate, understand, communicate: How developers locate performance bugs. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. ACM/IEEE, Oct 2015.

[4] C. Berka, D. J. Levendowski, M. N. Lumicao, A. Yau, G. Davis, V. T. Zivkovic, R. E. Olmstead, P. D. Tremoulet, and P. L. Craven. Eeg correlates of task engagement and mental workload in vigilance, learning, and memory tasks. *Aviation, space, and environmental medicine*, 78(5):B231–B244, 2007.

[5] J. T. Biehl, M. Czerwinski, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1313–1322, New York, NY, USA, 2007. ACM.

[6] S. Busechian, V. Ivanov, A. Rogers, I. Sirazitdinov, G. Succi, A. Tormasov, and J. Yi. Understanding the impact of pair programming on the minds of developers. In *Proceedings of the 40th International Conference on Software Engineering New Ideas and*

Emerging Results - ICSE-NIER '18, pages 85–88, 2018. Exported from https://app.dimensions.ai on 2019/02/24.

[7] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[8] C. Chen, W. Tao, and K. Zhang. Continuous awareness: A visual mobile approach. *Journal of Visual Languages & Computing*, 24(5):390 – 401, 2013.

[9] M.-Y. Chen, C. Chen, S.-Q. Liu, and K. Zhang. Visualized awareness support for collaborative software development on mobile devices. *International Journal of Software Engineering and Knowledge Engineering*, 25(02):253–275, 2015.

[10] B. Chu and K. Wong. Towards evidence-supported, question-directed collaborative program comprehension. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06, Riverton, NJ, USA, 2006. IBM Corp.

[11] I. D. Coman, P. N. Robillard, A. Sillitti, and G. Succi. Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software*, 91:124 – 134, 2014.

[12] R. Conradi and A. Wang. Empirical methods and studies in software engineering, experiences from esernet. *Lecture Notes in Computer Science*, 2765, 01 2003.

[13] C. Cook, W. Irwin, and N. Churcher. A user evaluation of synchronous collaborative software engineering tools. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 6 pp.–, Dec 2005.

[14] S. D'Angelo and A. Begel. Improving communication between pair programmers using shared gaze awareness. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6245–6290, New York, NY, USA, 2017. ACM.

[15] L. Deimel and J. Naveda. Reading computer programs: Instructor's guide and exercises educational materials cmu. Technical report, SEI-90-EM, 1990.

[16] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 241–248, Sep. 2005.

[17] R. DeLine, A. Khella, M. Czerwinski, and G. Robertson. Towards understanding programs through wear-based filtering. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 183–192, New York, NY, USA, 2005. ACM.

[18] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work*, CSCW '92, pages 107–114, New York, NY, USA, 1992. ACM.

[19] T. Dyba, B. A. Kitchenham, and M. Jorgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, Jan 2005.

[20] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, Dec. 1992.

[21] C. Gutwin and S. Greenberg. Workspace awareness for groupware. In *Conference Companion on Human Factors in Computing Systems*, CHI '96, pages 208–209, New York, NY, USA, 1996. ACM.

[22] J. E. Hannay, T. Dybå, E. Arisholm, and D. I. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110 – 1122, 2009. Special Section: Software Engineering for Secure Systems.

[23] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 3–9, New York, NY, USA, 1992. ACM.

[24] A. Jbara and D. G. Feitelson. How programmers read regular code: A controlled experiment using eye tracking. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 244–254, May 2015.

[25] P. Jermann and M.-A. Nüssli. Effects of sharing text selections on gaze cross-recurrence and interaction quality in a pair programming task. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1125–1134, New York, NY, USA, 2012. ACM.

[26] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[27] M. Lanza, L. Hattori, and A. Guzzi. Supporting collaboration awareness with real-time visualization of development activity. In *2010 14th European Conference on Software Maintenance and Reengineering*, pages 202–211, March 2010.

[28] S. Letovsky. Cognitive processes in program comprehension. In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, pages 58–79, Norwood, NJ, USA, 1986. Ablex Publishing Corp.

[29] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke. On the comprehension of program comprehension. *ACM Trans. Softw. Eng. Methodol.*, 23(4):31:1–31:37, Sept. 2014.

[30] J. I. Maletic and H. Kagdi. Expressiveness and effectiveness of program comprehension: Thoughts on future research directions. In *2008 Frontiers of Software Maintenance*, pages 31–37. IEEE, 2008.

[31] S. P. Marshall. Method and apparatus for eye tracking and monitoring pupil dilation to evaluate cognitive activity, 1999.

[32] K. Maruyama, T. Omori, and S. Hayashi. A visualization tool recording historical data of program comprehension tasks. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 207–211, New York, NY, USA, 2014. ACM.

[33] A.-L. Mattila, P. Ihantola, T. Kilamo, A. Luoto, M. Nurminen, and H. Väätäjä. Software visualization today: Systematic literature review. In *Proceedings of the 20th International Academic Mindtrek Conference*,

AcademicMindtrek '16, pages 262–271, New York, NY, USA, 2016. ACM.

[34] H. A. Müller, K. Wong, and S. R. Tilley. Understanding software systems using reverse engineering technology. In *Object-Oriented Technology for Database and Software Systems*, pages 240–252. World Scientific, 1995.

[35] C. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 68:046116, 11 2003.

[36] J. T. Nosek. The case for collaborative programming. *Commun. ACM*, 41(3):105–108, Mar. 1998.

[37] M. Nosál', M. Sulír, and J. Juhár. Source code annotations as formal languages. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 953–964, Sep. 2015.

[38] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1):5:1–5:58, Jan. 2018.

[39] M. P. O'Brien. Software comprehension - a review & research direction. TechnicaReport UL-CSIS-03-3, University of Limerick Ireland department of Computer Science & Information Systems, Nov. 2003.

[40] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood. Using developer activity data to enhance awareness during collaborative software development. *Computer Supported Cooperative Work (CSCW)*, 18(5):509, Oct 2009.

[41] R. Patnayakuni, A. Rai, and A. Tiwana. Systems development process improvement: A knowledge integration perspective. *IEEE Transactions on Engineering Management*, 54(2):286–300, May 2007.

[42] S. Pietinen, R. Bednarik, T. Glotova, V. Tenhunen, and M. Tukiainen. A method to study visual attention aspects of collaboration: Eye-tracking pair programmers simultaneously. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08, pages 39–42, New York, NY, USA, 2008. ACM.

[43] S. Pietinen, R. Bednarik, and M. Tukiainen. An exploration of shared visual attention in collaborative programming. *PPIG*, 05 2009.

[44] S. Pietinen, R. Bednarik, and M. Tukiainen. Shared visual attention in collaborative programming: A descriptive analysis. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '10, pages 21–24, New York, NY, USA, 2010. ACM.

[45] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 255–265, Piscataway, NJ, USA, 2012. IEEE Press.

[46] D. D. Salvucci and J. H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00, pages 71–78, New York, NY, USA, 2000. ACM.

[47] I. Schröter, J. Krüger, J. Siegmund, and T. Leich. Comprehending studies on program comprehension. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 308–311, May 2017.

[48] A. Seriai, O. Benomar, B. Cerat, and H. Sahraoui. Validation of software visualization tools: A systematic mapping study. In *2014 Second IEEE Working Conference on Software Visualization*, pages 60–69, Sep. 2014.

[49] Z. Sharafi, Y.-G. Guéhéneuc, and Z. Soh. A systematic literature review on the usage of eye-tracking in software engineering. *Elsevier Journal of Software and Information Technology (IST)*, 07 2015.

[50] K. Sharma, P. Jermann, M.-A. Nüssli, and P. Dillenbourg. Understanding collaborative program comprehension: Interlacing gaze and dialogues. *Computer-Supported Collaborative Learning Conference, CSCL*, 1:430–437, 01 2013.

[51] J. Siegmund. Program comprehension: Past, present, and future. In *2016 IEEE 23rd International Conference on Software Analysis,*

Evolution, and Reengineering (SANER), volume 5, pages 13–20, March 2016.

[52] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 378–389, New York, NY, USA, 2014. ACM.

[53] A. Sillitti, G. Succi, and J. Vlasenko. Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1094–1101, June 2012.

[54] R. Stein and S. E. Brennan. Another person's eye gaze as a cue in solving programming problems. In *Proceedings of the 6th International Conference on Multimodal Interfaces*, ICMI '04, pages 9–15, New York, NY, USA, 2004. ACM.

[55] M. . Storey. Theories, methods and tools in program comprehension: past, present and future. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 181–191, May 2005.

[56] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software development: A survey and a framework. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 193–202, New York, NY, USA, 2005. ACM.

[57] M. Sulír, M. Nosáľ, and J. Porubän. Recording concerns in source code using annotations. *Computer Languages, Systems & Structures*, 46:44 – 65, 2016.

[58] H. van der Meulen, P. Varsanyi, L. Westendorf, A. L. Kun, and O. Shaer. Towards understanding collaboration around interactive surfaces: Exploring joint visual attention. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16 Adjunct, pages 219–220, New York, NY, USA, 2016. ACM.

[59] A. von Mayrhauser and A. M. Vans. From program comprehension to tool requirements for an industrial environment. In *[1993] IEEE Second Workshop on Program Comprehension*, pages 78–86, July 1993.

[60] J. Whitehead. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[61] C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 38:1–38:10, New York, NY, USA, 2014. ACM.

[62] E. Ye, X. Ye, and C. Liu. Teamwatch: Visualizing development activities using a 3-d city metaphor to improve conflict detection and team awareness. *PLOS ONE*, 13(3):1–27, 03 2018.

[63] S. Yusuf, H. Kagdi, and J. I. Maletic. Assessing the comprehension of uml class diagrams via eye tracking. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 113–122, June 2007.

[64] I. Zayour and A. Hamdar. A qualitative study on debugging under an enterprise ide. *Information and Software Technology*, 70:130 – 139, 2016.

List of Figures

List of Tables

Appendix A

Evaluation of collaborative visualisation tools

The green colour refers to the availability of the feature in the approach. The assessment is based on the features that are included in each approach. For example the tools of desktop visualization are various and not all the tools offer the same features. So in this case, a feature is considered available, if it is present in one of the selected tools in this approach (best case is taken).

Table A.1: Evaluation of collaborative visualisation tools

Evaluation features		Attention shared by collaborative visualization tool			Attention shared by IDE plud-ins					Face to face collaboration
		Desktop visualization tool	**Mobile visualization tool**	**Interactive surface visualization tool**	**Annotation**	**Eye tracking**	**Text selection**	**Wear based filtering**	**Interaction data**	**Face to face collaboration**
Time [56]	Present									
	Recent past									
	Historical									
Kind of view [56]	Annotation									
	Graph view									
	Statistical view									
Navigation view [56]	Overview/ detailed									
	Coupled view									
	Zoom-able view									
Effectiveness [56]	Scalability									
	Interoperability									
	Learning									
Efficiency [29, 57, 6]	Productivity									
	Mental workload									
	Cognitive workload									
	Distraction elimination									

Evaluation features		Attention shared by collaborative visualization tool			Attention shared by IDE plud-ins					Face to face collaboration
		Desktop visualization tool	**Mobile visualization tool**	**Interactive surface visualization tool**	**Annotation**	**Eye tracking**	**Text selection**	**Wear based filtering**	**Interaction data**	**Face to face collaboration**
Expressiveness [21, 29, 40] [45, 57, 62]	Change detection									
	Conflict detection									
	Mental model									
	Familiarisation with the project									
	Intention									
	Activity									
	Identity									
	Location									
	Sphere of influence									

Appendix B

Summarisation of the literature review papers

Table B.1: Summarisation of the literature review papers

Paper	**Author**	**Year**	**Environment**	**Type of participants**	**Task type**	**Methods to evaluate the study**
[44]	S. Pietinen et al.	2010	Industrial like environment	Professionals	Collaborative programming	Eye tracking + verbal protocol
[13]	C. Cook et al.	2005	N\A	Professionals	Collaborative programming	CAISE based CSE tools
[57]	M. Sulír et al.	2016	Industrial + controlled environment	Professionals+ students	Program comprehension	Observation (Think aloud method)
[10]	B. Chu et al.	2006	Controlled environment	Students	Program comprehension	video+ sound are recorded+ written notes are analysed
[64]	I. Zayour et al.	2016	Industrial environment	Professionals	debugging	Observation+ interview
[5]	J. T. Biehl et al.	2007	Controlled environment	Professionals	Collaborative programming	Survey + interview + pre and post observation
[14]	S. D'Angelo et al.	2017	Industrial environment	Professionals	Collaborative programming	Eye tracking + video analysis + interview
[45]	T. Roehm et al.	2012	Industrial environment	Professionals	Program comprehension	Observation (think aloud method) + interview
[3]	S. Baltes et al.	2015	Industrial environment	Professionals + students	Debugging	Profiling tool + Observation (think aloud method) + interview + audio record + video of the screen +log of various navigations

Paper	Author	Year	Environment	Type of participants	Task type	Methods to evaluate the study
[42]	S. Pietinen et al.	2008	Industrial environment	Professionals	Collaborative programming	Eye tracking + screen capture with cursor of the test programs + facial videos of users
[29]	W. Maalej et al.	2014	Industrial environment	Professionals	Program comprehension	Survey + observation + interview
[62]	E. Ye et al.	2018	Controlled environment	Professionals + students	Collaborative programming	Video recording + chat logs + survey + CVS repository
[43]	S. Pietinen et al.	2012	Industrial environment	Professionals	Collaborative programming	Eye Tracking
[16]	R. DeLine et al.	2005	Controlled + industrial environment	Professionals	Program comprehension	Interview + observation
[40]	I. Omoronyia et al.	2009	Controlled environment	Students	Collaborative programming	IDE interactions
[6]	S. Busechian et al.	2018	Controlled environment	Students	Collaborative programming	Brain activity of the user
[54]	R. Stein et al.	2004	Controlled environment	Professionals	Debugging	Eye tracking + video recording + observation (think aloud method) + audio record
[8]	M. Lanza et al.	2010	Controlled environment	Students	Collaborative programming	Syde
[17]	R. DeLine et al.	2005	Industrial environment	Professionals	Program comprehension	Observation(think aloud method) + log of code information

Paper	Author	Year	Environment	Type of participants	Task type	Methods to evaluate the study
[53]	A. Sillitti et al.	2012	Industrial environment	Professionals	Collaborative programming	PROM
[32]	K. Maruyama et al.	2014	N\A	N\A	Program comprehension	Notes + recorded actions
[27]	C. Chen et al.	2013	Controlled environment	Professionals + students	Program comprehension	Average correctness+ + completion time
[58]	H. van der Meulen et al.	2016	Controlled environment	Students	Collaborative programming	Eye tracking
[9]	M.-Y. Chen et al.	2015	Controlled environment	Students	Program comprehension	Eye tracking + video recording + interview
[50]	K. Sharma et al.	2013	Controlled environment	Students	Program comprehension	Eye tracking + observation
[25]	Jermann et al.	2012	Controlled environment	Students	Collaborative programming	Eye tracking + video recording + logs of selection + audio record

Appendix C

Types of awareness information

Table C.1: Types of awareness information inspired by [62]

Tool/ Model	**Type**	**Awarness source**	**Awarness visualization**	**Awarness filter**	**Integrated communication**	**Type of information**
Team-WATCH **[62]**	Standalone	Version control repository, local workspace and issue tracking system	3D visualization for each developer Standard view for all team member	Awareness information are chosen by the developer	N\A	Historical and real time information
Team radar mobile **[8, 9]**	Standalone	Local workspace	2D visualization for each developer and team leader	Awareness information are chosen by the developer	Phone call message and email	Historical and real time information
CRI model **[40]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	Awareness information are chosen by the developer	N\A	Historical and real time information
FAST-Dash **[5]**	Standalone	Version control repository, local workspace	2D standard view for all team member	N\A	Asynchronous communication with annotations	Historical and real time information
Team tracks **[16]**	Standalone	Local workspace	2D standard view for all team member	N\A	N\A	Historical and real time information
Code-Forest **[32]**	Standalone	CodeForest automatically records a user's actions	3D visualization for each developer	Awareness information are chosen by the developer	Asynchronous communication with annotations	Historical information (user action +annotations)
Recording concerns using annotations **[57]**	Standard IDE of java language	Annotation are written by the user	Annotations are included in the code	Developer searches for specifics annotation with search function	Asynchronous communication with annotations	Historical information

Tool/ Model	**Type**	**Awarness source**	**Awarness visualization**	**Awarness filter**	**Integrated communication**	**Type of information**
Scamp and syde **[27]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	N\A	N\A	Real time information
Pollinator **[64]**	Eclipse plug-in	Version control repository and local workspace	2D standard view for all team member	N\A	Asynchronous communication	Historical and real time information
CAISE based CSE tool **[13, 32]**	Eclipse plug-in	Version control repository and local workspace	2D visualization for each developer	Awareness information are chosen by the developer	Direct communication	Historical and real time information
Wear based filtering **[17]**	Eclipse plug-in	Version control repository and local workspace	2D standard view for all team member	N\A	Asynchronous communication with annotations	Historical and real time information
Collaborative programming **[53, 42, 6] [3, 64, 10] [14]**	N\A	Direct communication	Shared visual attention or shared display	N\A	Direct communication	Real time information
Eye tracking **[44, 14] [54, 43]**	Eclipse plug-in	Eye Tracker shared eye gaze	2D standard view for all team member	N\A	Asynchronous or synchronous communication	Historical or real time information
Text selection **[25]**	Eclipse plug-in	Selection of a part of the code	2D standard view for all team member	N\A	Synchronous communication	Real time information