Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Software Engineering

Learning as an Outcome of Code Review at Capgemini

Reem Al Jaamil Alrashid

Matrikelnummer: 4821928

reemalrashid@zedat.fu-berlin.de

Betreuer und Erstgutachter: Prof. Dr. L. Prechelt

Zweitgutachterin: Prof. Dr. C. Müller-Birn

Berlin, 8.12.2017

Abstract

Code Review is known to be an efficient technique for finding defects in code. The technique also has other benefits, including the promotion of knowledge sharing, community building, and maintaining code quality. In order to improve this process and embrace its outcomes, it is necessary to better understand it, identify the factors influencing it, and study its side effects. Grounded theory study is performed to clarify the human and organizational factors, that influence the code review process. The study is based on interviews with software developers from different teams at Capgemini. These interviews provided insights into the process and the influencing factors behind some of the decisions taken during the process. The findings of this study are summarized as seven hypotheses. The results show the importance of the learning outcomes through the results of the code review process and on the developers. Moreover, learning is also influenced by the relationship between the developers and their knowledge. Based on those findings, some suggestions are given to improve the code review process.

Declaration of Authorship

I certify under penalty of perjury, that the present work has been produced by me independently, all tools and sources are used as indicated and the work has not been submitted to any other institution for consideration.

Berlin, December 8, 2017 _____

Acknowledgments

I would like to express my sincere gratitude to my supervisors Prof. Dr. Lutz Prechelt and Prof. Dr. Claudia Müller-Birn for their support, cooperation, and for providing me with all the necessary facilities for this thesis. I have the utmost appreciation for Prof. Dr. Claudia Müller-Birn for the constant encouragement and help she has provided me during the last two years. I am also very thankful to Helena Barke for her sincere and valuable guidance and encouragement extended to me.

I would also like to extend my great appreciation to Lukas Birn for his professional expertise and his willingness to support me with his knowledge, valuable suggestions and discussions during the research. Many thanks to all the participants for the time and effort they donated for the interviews.

Last but not least, I would like to thank my lovely family for everything. My mother for being my first teacher, my father for being my first supporter, my sisters and brothers-in-law for always being my role models. Many thanks to my best friends for their moral support and the amusing time.

Table of contents

List of figures

List of tables

1 Introduction

1.1 Capgemini

Capgemini is a consulting and IT services company founded in 1967 and headquartered in Paris. The company is represented in more than 40 countries and has more than 180,000 employees worldwide. *Capgemini* Deutschland, headquartered in Berlin, is responsible for business in Central Europe. The company has four main business areas; Application Services, Consulting Services, Outsourcing Services, and Local Professional Services. This research took place in Berlin in the field of Application Services. The services of this company are diverse, in Germany they are strongly represented in the automotive industry. In addition, conception, programming and operational management are offered.

1.2 Code Review

''To err is human''[1], it is part of human nature to make mistakes and to be unable to sufficiently review or read our own work; we need someone to help us see our work clearly. Code review, also known as inspection, has emerged because developers, often do not notice their own errors [1]. Code review is the process of examining a source code and finding the mistakes it contains. The first process of this kind is the Fagan Inspection, which was developed by Michael Fagan in 1976 [2]. It is an intensive process, and it requires multiple meetings with the participants. It is a team effort with at least 4 participants; author, two reviewers and moderator, and consists of the following five phases [3]:

1. Overview: The author presents the material intended for review and the goals of the inspection.
2. Preparation: The reviewers review the implementation of the component.
3. Inception meeting: In this stage the search for defects begins and the team raises the issues with the component. The moderator leads the meeting.

1. "To err is human, to forgive is divine." Alexander Pope

4. Rework: The author revises the component, in order to resolve the defects, found during the inspection meeting.
5. Follow-up: The moderator checks the quality of the rework, verifies that all the defects are fixed and that during the rework phase no new defects have emerged.

This process ensures the uncovering of defects in the examined source code, but, at the same time, it comes with a high cost, namely time. For this reason, many teams have avoided adopting it [4]. They have, instead, adopted some of the new and faster inspection processes that have been emerging. In this research the term code review (CR) will be used to refer to those processes, which are formal and written, but not so painstaking as the Fagan Inspection. In these processes there are two participants; the author and the reviewer. The author writes the code and then sends it to the reviewer. The reviewer reads the code. If mistakes are found, the reviewer sends the code back to the author with suggestions for changes. The reviewer decides when the code is ready to be committed to the team's codebase. The SmartBear study of Cisco Systems found that heavyweight inspection does not find more defects than the lightweight CR, although it takes more time [5]. There are different kinds of CR such as [6]:

- **Over-the-shoulder:** the developer walks the reviewer through the code, and the reviewer asks questions if they arise. Then the developer documents the defects and fixes them.
- **Email pass-around:** The developer sends the code to the reviewers and receives the results of their examination via mail.
- **Pair Programming**: Two developers write the code together, one at a time. They discuss and review the code continuously.
- **Tool-assisted:** Authors and reviewers use a special tool in the code review process, which organize the process by collecting, transmitting and displaying files, and giving product managers and administrators some control over the workflow.
  The reviewer during the CR process does not test if the code works or not; they rather search for mistakes that affect the quality of the code. Potential mistakes are [6]:
  - performance-related; such as, checking if there is a better way to write a particular piece of code; or if there is a better algorithm or approach

that could be used to do the expected task more efficiently.

- maintainability- and/or readability-related; where the reviewer checks if the names of fields, variables, parameters, methods, and classes reflect what they really represent They also check if the code is comprehensible, the tests are clear and if they cover all of the intended cases. Additionally they check whether there are cases that have not been considered, whether the complex sections have been documented and whether clear comments have been added.
- functionality-related; the reviewer checks, if the code does what it is supposed to do and checks if the correctness test really tests the requirements. The reviewer also looks for small bugs, such as using the incorrect variable for a verification or using "and" instead of "or".
- design-related; the reviewer checks if the author has followed the coding standards and the guidelines of the project, and they also check if the code is in the right place.

  In addition, there are security, structure, logic, redundancy, and many other issues that should be considered during the CR process. Furthermore, another study has found that the kind of problems spotted during the CR process cannot be found by later testing processes or by the field usage because they do not exhibit visible effects on the execution behaviour [1].

Moreover, CR enables finding mistakes in the early stages, which makes it easier and cheaper to remove them. In 1986, experts at IBM realised that programmer's work was cut by 85% because major defects in the code were found through the earlier stage of inspection instead of during later stage testing [7]. Later, in 1992, they also found that around 30 hours of maintenance work were saved for every hour spent in inspection [8]. Those findings imply that CR has constantly proven its competence in accelerating and streamlining the process of software development. In addition, a large number of developers participate in CR and spend circa six hours per week in this process [9]. Therefore, increasing the effectiveness and the benefits of the CR process would improve the development productivity in the company [1].

CR also helps to plan and organise the development process of the software because through CR the effort needed to complete a specific part of a code can be better estimated [9]. As the reviewer shares the knowledge of the

author, which is related to this specific task, more information will be used for the final estimate, and that implies a stronger and more reliable estimation [9]. Additionally, CR does not only find defects, improve the maintainability of the software, and help organize and improve the project, but it also helps share knowledge among developers, build relationships, and bring the team members closer to one another[1]. Since defect detection is the centre of most of the CR researches, the other benefits of CR research have not received much attention, even though they were important for the developers [1]. And according to Bacchelli and Bird [10], it is advisable to embrace the unexpected benefits of CR. They advised that code review policies should be guided with the explicit intent to improve those outcomes, instead of trying to refocus code reviews on finding defects. Because of the aforementioned points the focus in this research is on these less discussed outcomes.

Many technical as well as non-technical factors affect the complicated process of CR [11], the code size, the priority of the code, the number of modified files, the relationship between the developers, and experience, to name some examples. Many researchers have studied the effects of technical factors on CR [1], but only few have considered the non-technical factors, even though the non-technical factors, such as organizational and personal factors, are better predictors for the outcome of the CR process [12]. For this reason, this thesis focuses on the non-technical factors, such as the relationship between the author and the reviewer.

1.3 Contribution

The goal of this research is to improve the CR process at *Capgemini*. The companies which carry out the CR process vary in the details of their processes [13]. Thus, it is important to understand the current process and identify improvement opportunities in order to enhance it. Additionally, it is necessary to evaluate the current process in order to be able to improve the process by incorporating suggestions.

At the beginning of the study there were many interesting questions, including the following:

1. How does the current process of CR at *Capgemini* function?
2. How do human factors affect the process and in which way?

3. How do developers feel if they receive a feedback full of corrections and change suggestions?
4. When does knowledge transfer happen and what is the best way to improve it?
5. How to arouse greater interest in CR?
6. What are the characteristics of a good reviewer?
7. When is a CR considered successful?

This study has researched information related to the aforementioned questions and studied how this information affects the CR process and its outcomes, and which factors lead to a delayed or unsuccessful review. After that, suggestions are proposed to improve the CR process.

1.4 Structure

This study is organized as follows: Section 2 describes the research method, Section 3 provides the results and Section 4 describes the improvement suggestions. Section 5 introduces related works and briefly compares them with this study. In Section 6 the limitations of this study are indicated. Finally, Section 7 provides directions for future work and concludes the thesis.

2 Research Method

In this research, elements of Grounded Theory Method (GTM) are applied. This section presents the GTM and describes its applied elements.

2.1 Grounded Theory

A Grounded Theory (GT) is a theory that is deduced to reveal theoretical principles about a phenomenon under study, GT was developed in 1965 by Glaser and Strauss [14].

A phenomenon can be anything related to people's lives, stories, behaviours, or about organizational functioning, social movements, or interactional relationships. Any phenomenon can be studied from different perspectives [15]. For example, when studying children in a school, one can study a phenomenon from the educational perspective where researchers can examine the learning process, though they can also study it from the psychological per-

spective, where they study group dynamics [15]. The chosen GT phenomenon should not have been studied in depth yet, or, the relationships between the concepts should still be unclear or undeveloped [15].

Grounded Theory allows the researcher to move from data to theory. One starts with studying a phenomenon and what is relevant to it, rather than beginning with a specific theory. Then the Grounded Theory is developed through a qualitative research method – which is the GTM. GTM uses a systematic set of procedures in order to develop the theory. It was developed by two sociologists, Barney Glaser and Anselm Strauss [14] after that, the methodology has been influenced by different schools of thought over the years. Therefore, there are different versions of GTM, in this study strategies of the GT are applied

2.2 Collecting the data

The GTM starts by collecting data about the phenomenon, in order to analysis it afterward. The data is collected as follows.

2.2.1 Theoretical Sampling

"Theoretical Sampling is sampling on the basis of concepts that have proven theoretical relevance to the evolving theory." [15]

There are a few important aspects to take into consideration before starting the procedure of data collection. Here e.g., Strauss and Corbin[15] emphasize the significance of choosing the appropriate type of data. Observations, documents, interviews, and combination of these types are different ways of conducting the grounded theory research and are applicable for this study. A qualitative grounded theory approach with semi-structured interviews was chosen as the optimal method to address the aim and objective of the project and to answer the research questions. A semi-structured interview is an open form of interview, where new ideas brought up by the interviewee during the interview are allowed, and the interviewer has a framework of aspects they want to explore.

One should also choose the group which will be studied [15], which are the interviewees in this research. Because this work is about CR, only partici-

pants of the CR process, authors, and reviewers were interviewed. In theoretical sampling, it is important to find representativeness of the concept in its different forms rather than finding a portion of a population to represent the entire population [15]. Because of that, only developers – as authors or reviewers – who participated in CR were interviewed as representative for the CR process. Developers with sufficient experience can better describe the process of CR and estimate its benefits. They, in addition, have experienced this process with different kind of developers. For that reason, chosen interviewees have participated in adequate number of CR, namely minimum one year of experience in this process. Moreover, it is essential to understand the process in different projects or teams, that is why the interviewees were chosen from three different projects.

2.2.2 Qualitative Interview Design

Because this research focuses on the human aspects and the organisation of the CR process, it was important to understand the theory in use, which the interviewees apply at work, rather than their espoused theory. Whereby espoused theories are what one believes themselves to believe in, despite the fact that they may not always act according to those beliefs[16]. And theories-in-use are the ideas that actually guide their daily actions [16]. For this reason, multiple questions addressed particular experiences or examples to gain information about the interviewee's theory in use. Moreover, exploratory verbs were used, such as "describe" and "tell me about", as Creswell suggested [17]. For instance, it is questionable who is a good reviewer. Are they the ones who send the code back with the minimum numbers of defect or are they the ones who explain a lot or..etc. So I asked the authors about their last experience with their favorite reviewer as following: *"Is there a developer whose code you like to review? Would you tell me about your last code review experience with them?"*, rather than asking directly about their opinion: " What is the reviewer's job, in your opinion?". The first question helps to understand who are actually the *good* reviewer in practice and what is actually expected from them, according to the authors. Due to the fact, that the authors are evoking experiences and their triggers from previous experiences. The second will let the interviewee only answer as they believe and not based on how they act.

Furthermore, all the questions were formulated in an open-ended form, as Creswell suggested[17] in order to seek more data. For that reason, ques-

tions were started by “how” or “what”. The word “why” was avoided because in qualitative research, the researcher is not trying to explain the occurrence of something, rather describe how it occurred [17]. Moreover, the semi-structured interviews help to discover aspects that were not considered while writing the interview questions, but they can still be considered in the analysis.

The questions were either in German or in English, the language of the conducted interview was decided by the interviewee. In this study citations used were either in English or were translated to the English. Moreover, the questionnaire is divided into four main parts. The first part is a boarding section, where an icebreaker question is asked [17] to make the interviewee feel more comfortable and the interviewer starts exploring the central phenomenon and would gain information about the CR process in the interviewee’s team [17]. The second part contains three questions about the CR style and the quality of the code. The third part has five questions, which handle the human aspects of CR, such as the relationship between the CR team members. The last section, which consists of three questions, is about conflicts and communication during the CR process. Every part has its own target data but any related data from any part of the interview was considered and noted, the target data is defined as follows:

- The second part: from this part reasons for doing CRs and expectation from this process were sought.
- The third part: Information about the team were expected from this part of the questionnaire. When is a developer considered a good reviewer or a good author and why? And finally how the knowledge of the team, project or the company affect the process of CR.
- The last part: In this section information about conflicts, misunderstandings and dissatisfaction was sought. How do the developers react in such situations and what strategies do they use to avoid them.

2.2.3 Pilot Tests

Three pilot-interviews were conducted so that the comprehensibility and validity of the questionnaire is guarantied. These interviews led to changes: in (1) the formulation/phrasing of the question. For instance, instead of asking the reviewer why they like to review the code of a specific developer, they have been asked about the last experience with them. (2) adding and re-

moving some questions and (3) reordering the questions. Those pilot-interviews also detected any difficulties in comprehending the questions and assured that the estimated time (30-45 Minutes) for one interview was sufficient.

To ensure there is no information loss and to make it possible to return to the data when needed, all the interviews were recorded and transcribed. I received the interviewees consent of participation and the use of the audio recorder during the interviews. Moreover, the anonymity of all participants was ensured. For transcribing *F4transkript* is used and some of Mayring tips of were considered [18]; Such as:

- Everything was completely and literally transcribed.
- Dialect is translated into German or English as much as possible, where the meaning of the sentence is retained.
- Unclear utterances are replaced with (...).
- Vocalisations of the interviewee supporting this statement or making it clearer (laugh or sign) are noted in brackets.

2.3 Analysis

2.3.1 Coding

Coding is the operation of breaking down the data, conceptualized it, and putting it back together in new ways [15]. It is considered as the central process for building theories from data. GTM as mentioned above is a qualitative research method and qualitative analysis means to classify objects of an area them into different categories. Categories consist of objects which share central features or characteristics [15]. One category at the beginning of this study was *Conflicts,* where the interviewee have described and talk about different conflicts situation or disagreement experiences. The objects of this category shared these characteristic disagreement, unpleasant experience or situation, obstacle for the process, to name some examples. The Objects of this category were "*Sprachbarriere*", "*Probleme*", "*Sachen die ich nicht so mag*", "*not comfortable*", for instance.

There are three different kinds of coding which were applied in this study, "open coding", "axial coding" and "selective coding" [15]:

- "Open Coding is the process of breaking down, examining, comparing, conceptualising, and categorising data."
- Axial Coding is a set of procedures whereby data are put back together in new ways after open coding"
- Selective Coding is the process to be made after going back and forth between open and axial coding. In this process the main category is selected, on which the research will focus. Other categories will be linked to it in a form of relationship, then those relationships will be validated.

2.3.2 Constant comparison

During constant comparison [19] one makes sure that all categories built are different from each other and that there is no category which is a subcategory of another one. Constant comparison links and integrates categories in such a way that all variations are recognized and ensures that there is no duplication in the categories and that the diversity of the data is shown. For instance, two categories "Project Knowledge" and "Experience" were merged into one category, namely, "Knowledge".

2.3.3 Memo-Writting

It is the process of recording thoughts and ideas during analysis, whereby notes can be written or drawn [15]. Memos are important to keep every analytical process documented and to be able to examine the theory. They are also useful in the constant comparison process to compare the definitions or the memo of the categories. Additionally, they help in the writing and presenting phase. Memos can be notes about the categories such as definition of a particular category or its characteristics, about the relationship between the categories and also about the logical relationship between the categories and their subcategories. The researcher must start writing memos from the beginning of the research, however, most of the memos will probably appear simple at that time. Then memos start to evolve throughout the research as memo writing must continue until project is concluded. Every memo should be dated, thus at the end of the research it will be possible through these memos to observe the development and the growth of the research and it will be easier to find a specific memo by date.

After every single interview, coding was done in order to find relevant

themes and concentrate on them in the next interview. The data was analysed using Atlas-ti software program. After the last interview there were 13 categories, such as Knowledge, Conflicts, Communication. Then coding continued to be done repetitively. After open coding, axial coding, constant comparison and going back and forth between these elements only 5 categories were left. These categories are presented in the coming section.

3 Results

In this section, the main results are presented. The results take the form of grounded hypotheses, since the methodology used is theory generating. The Subsection 3.1 gives an overview about the interviewee's team. In the remaining subsections the final results of the analysis are presented.

3.1 Overview

Each team consists of 3 to 10 developers. For this study, 7 developers from 4 different projects were interviewed. Three interviews were pilot-interviews, where the developers from three different teams were chosen. For the other four interviews, the developers were from three different projects. In three teams (IDs A, C, and D in Table 1) the developers were the authors of the code in some CR processes, and in other processes they were the reviewers. In other words, each has experience of both roles. In contrast reviewers in team B did not work as developers. Interviewees from the teams A, C, and D were asked about their experiences in both roles, as authors and as reviewers. Unfortunately, no author from team B was interviewed, due to the lack of time available due to the lack of time available to complete the research.

Table 1. Teams overview

Teams	Interviewees
Team A	INT1, P-INT1
Team B	INT2, INT4, P-INT2
Team C	INT3
Team D	P-INT3

P = Pilot Interview

The CR process begins when the author submit their written code to be reviewed. CR may consist of one or several rounds. Each round is an exchange of code and feedback between the author and the reviewer– where the reviewer checks the submitted code and sends their suggestions for improvement. The last round takes place when the reviewer approves the code and accepts to merge it with the codebase. Moreover, a survey based on the responses from more than 550 software developers, testers, IT-operations professionals, and business leaders from more than 30 different industries [20] showed that 75% of respondents use a combination of the aforementioned CR methods. Similarly, the teams at *Capgemini* use a combination of these methods. The main method is "tool-assisted", where one of the code review tools is primarily used such as "Gerrit", and when necessary other methods will be implemented, such as: "over-the-shoulder" or "email pass-around", in addition meeting to discuss the suggested changes is also possible.

3.2 Hypothesis Generation

This study showed that learning was an important outcome of CR which in turn affected the CR process and the results. According to Bacchelli and Bird [10] it is recommended to embrace the unexpected outcomes of CR. Rather than trying to refocus code reviews on finding bugs, they advise that code review policies should be guided with the explicit intent to improve outcomes such as, increased learning. Thus, the main focus of this study is *learning*. Learning from CR can be influenced by many factors. In this study two factors are presented and explained. Furthermore, the results of learning on the

project, on the team, and on the developers themselves are introduced. In order to refer to statements from the interviews, the interviewees ID's from table 1 are used as subscript.

3.2.1 Learning

Learning or knowledge-sharing is an established benefit of the CR process [1][6]. Through CR the project knowledge and every team member's knowledge is shared across the team. Consequently, the whole team is improved and therewith the quality of the code in the long term also improves.

Moreover, developers always seek learning from each other. According to AIFORSE Community [21], developers spends on average about 3h looking for information from other colleagues. Mainly because they cannot find what they are looking for anywhere. The article also showed that approximately 4h daily per developer are spent on this learning process and about one and half hour from them are actually wasted. That implicates that learning is actually requisite. Through CR the learning and knowledge sharing processes could be intended, which probably would save the aforementioned wasted time.

Developers at *Capgemini* look at learning from different perspectives. For experienced developers, it was important to teach and mentor other developers;

> "*Whereby the reviewer should play the role of a mentor and not just highlight the issues with respect to the code…*"$_{\text{INT4}}$[2].

For other developers, they found that CR enabled them to discover their own mistakes and learn from them as well as from the feedback of the reviewer. Thereby, they discover their own weaknesses and try to overcome them. One interviewee stated:

> "*The author learns from the feedback and avoid doing the same mistake in the future…*"$_{\text{INT1}}$.

2. The subscript given at the citations from the interviews refer to the interviewee ID from table_1. English translations are used for citations when the interviews were done in German. The full interviews are available in: "https://github.com/Reem00/Interviews-Bachelor-Thesis"

Besides, authors learn from the review process how to review a code and how to write a good review. When they make a mistake, they not only avoid it in the future, but they also make sure when reviewing a code that similar mistakes do not occur elsewhere.

"*To consider this mistake in the future tasks and to learn from it*"INT3

Moreover, interviewees believed that even the reviewer benefits from the CR process, because they found that CR forces the reviewer to think of a better way to write a piece of code when they do not like the written one, or to think of a solution to the problem that they have found. In that way they practise and train themselves:

"*I must think of a better way to implement it.*"INT2

The developers also agreed that it is useful to see code from different developers in order to see different code styles or to learn new methods:

"*you see how other people code then you get different view*"INT2

Finally, by reviewing code from more experienced developers, the reviewer gets to see what good code looks like, recognize what is expected of their code, and learn from other experts:

"*New colleagues can learn from the pretty good code*"INT1.

Despite the developer's different perspectives of learning, all developers consider learning an important advantage of CR [1]. The developers at *Capgemini* share the same opinion; one interviewee even described the CR process as a "learning process"INT1. They considered the CR process successful, not only when defects were found, but also when they gained new knowledge in the process. One developer said:

> "I was not disappointed yet by any CR process. I often take something from the CR process, which I can use in a further development, something like a guideline, or like a new and easier or clearer way to write Java code…"INT1.

Moreover, developers were able to measure their progress through feedback, and reviewers felt more confident when reviewing a code and find-

ing something to improve or to correct:

> *"Then you are happy about it. So, I'm not happy that they made the mistake, I'm just happy about the fact that I found something. I took out a risk or a mistake"* INT3

Moreover, learning also has its own results and reflections on the developers. After a pleasant CR process, reviewers and authors felt satisfied either because they learnt something or because they taught the author something:

> *"And then I'm also happy when I can teach other people something"* INT3,

> *"On the other hand, I learnt from it"* INT1

Otherwise, conflicts might occur. Even by further review processes, reviewers felt either proud when they read a good code, because they know that the author has learnt something and have developed their skills from previous feedback, or conversely they felt upset and disappointed when authors repeated the same mistakes and did not learn from their mistakes:

> *"They never learn"* P-INT1.

Hypothesis (1): When developers learn something from the code review process, then they are satisfied with it, independently of whether they faced conflicts during the process or not.

Hypothesis (2): Due to self-estimation during the learning process (by correcting or being corrected through others) the confidence of the team members increases.

3.2.2 Feedback

During the CR process the reviewer checks the code, and when they find a mistake or something that could be improved, they send feedback to the author. There is also positive feedback, when the reviewer praises the author's work and confirms the correctness of their code. When demanding a change in the code, the reviewer sometimes justifies their change suggestions by giving arguments and sometimes they do not. The explanations are sometimes con-

cise and sometimes detailed. The methods of sharing arguments and explaining change suggestions differ according to the case. Even the way of formulating or phrasing the comments differs from one CR to another.

All of the teams, as mentioned before, use a tool for the CR, but they use other kinds of communication to discuss the feedback if necessary. In other words, the reviews are always recorded in the used tool, but if the reviewer relies on other types of communication to discuss the feedback, then that depends on the particular case. Sometimes the reviewer "*go[es] through the code with the author*"INT2 and explains to them, where changes could be made, how, and why. Sometimes they "*sit together*"INT1 and discuss the desired changes. In this case, the reviewer may "*draw something*"P-INT2 in order to explain their point. If it is not possible to meet in person, the reviewer and the author could *"meet via Skype"*P-INT2 instead.

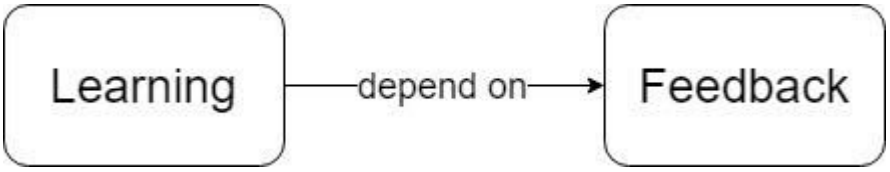

Fig. 1. Learning-Feedback

Learning is mostly earned from the feedback of the CRs, as the developers in this stage discuss the changes:

> "*the developer opens his code and we go through it and I said I do not think it is a right way. We can do it like this then we have some discussion about it… and the colleague has learnt out of it*"INT2

From the feedback stage the developers learn, at the minimum, about their mistakes and make sure the things they wrote were correct. However, to understand and learn more, the reviewer has to relay the necessary information to the author.

Hypothesis (3): Learning from the code review is dependent on the feedback.

The two following factors affect the feedback and influence its usefulness and effectivity.

3.2.3 Knowledge

There are different types of knowledge that the developer could have; their previous education, the experience that they have gained in different fields, and their knowledge of the company, team, and the specific project which they are working on. Obviously, knowledge is an important factor for learning. Because learning is either giving or gaining knowledge. Nevertheless, the author's knowledge affects the process of CR, specifically, the reviewer's feedback.

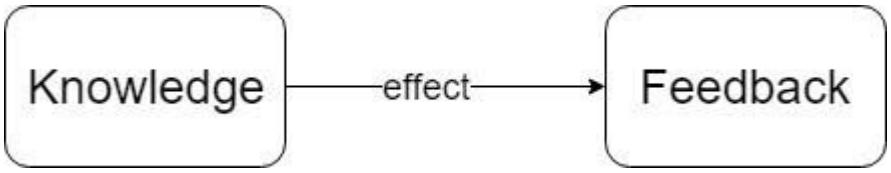

Fig. 2. Knowledge-Feedback

The reviewer takes into consideration the author's knowledge while doing the CR; they have in mind what the author has studied or worked with before, whether they are new to the project or to the team, whether they know the guidelines and the standards of the project etc. For instance, when a new developer joins the team, the reviewer is expected to teach them the standards of the project and the general guidelines. During the feedback, the reviewer should explain to the author where they have made mistakes, justify the change suggestions, and always refer to the standards or the guidelines, which the author is expected to follow:

> "*For new colleagues one has to pay more attention to give them the right information, so that they learn for the future*"INT1

> "*I can understand that freshers do not have much experience with the project … I ask questions about the code so I can understand and help them*"INT2.

Moreover, developers with less experience or less knowledge about the project tend to make more mistakes, not only due to lack of knowledge but also confidence issues. They make mistakes such as typo mistakes or documentation-related mistakes, because they are confused and stressed at the beginning. One interviewee explained:

*"people who are very new to the project are those who are not so comfortable with respect to coding the complex changes, they – most of them – end up making mistakes and it could be you know, very small mistakes, like documentation…"*INT2

Some reviewers were aware of that, and knew that the words they use in the review have impact. Hence, they were more careful when writing feedback and tried to be kind and encouraging to avoid destroying the author's self-confidence. Another issue related to the author's knowledge is when a developer is asked to review code by a more experienced developer. The reviewer then feels *honoured* to be considered capable of reviewing and checking the code of this experienced developer. That motivates them to do their best while reviewing and to write a good feedback.

*"I had that task to do review the code of a very experienced colleague in my opinion. Then, I first thought, why should I look at the code, what should I find there ... but that had to be done anyway. I sat down, looked at it and then had a few minor comments, but that was almost my very first experience with it and that was fun at the end."*INT3

Hypothesis (4): Author's knowledge affects the feedback of the code review.

To conclude, reviewers wrote their feedback based on the author's knowledge. When the author was a less experienced developer or was relatively new to the team, then the reviewer tried to transfer and share their knowledge with the author. Some reviewers were aware of the fact that new developers in the team were confused and tend to make more mistakes, so the reviewers help them counter this problem. In addition, reviewing a code, written by a more experienced developer, gratified the reviewers and motivated them to write good feedback.

3.2.4 Relationship

The relationship among colleagues plays an important role in the process of CR. Some reviewers have to review codes written by developers, whom they have never seen or heard of, and sometimes they are asked to review their friends' code. There are cases in which the author and the reviewer have worked together before and so they know each other's working style and

habits. Moreover, the author and the reviewer could be working together on-site or offsite.

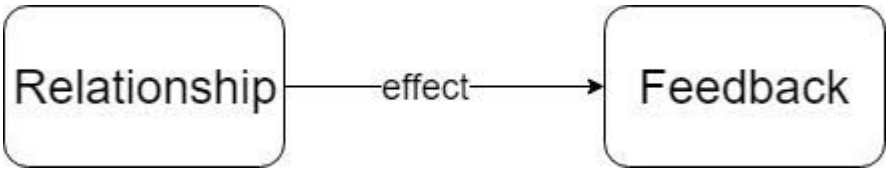

Fig. 3. Relationship-Feedback

The interviews show that these different kinds of relationships affected the CR process, specially the feedback part. Reviewers always kept in mind the character and experience of the person reading the comments, a developer whom they know or have met. That affected the way they wrote the comments, their choice of words, and also their style – be it formal or informal. They tend to pay more attention to their words and phrasing when they do not know the author well. This can be due to the fact initially in collaborations there is a fear of vulnerable situations, since the reviewer does not know anything about the author's acceptance to criticism. One reviewer explained:

> *"I wrote a bit more formal, at the beginning you do not know how your colleague will react "* INT1

In addition to being less formal when writing to a well-known author with a history of cooperation with the reviewer, both developers, author and reviewer, tend to be more direct with each other, have less difficulties criticising each other, and have less intense discussions. One of the interviewees said:

> *"I enjoy working with colleagues whom I know. Because there are no limits and you are just honest. You can easily say: "Sorry, but the code is totally bad" and then you get a reply, such as "yes I was totally tired yesterday" or "I had to go home urgently, that's why I checked out the code like that". That comes with time."* P-INT2

Hypothesis (5): Good relationships lead to more useful feedback.

Good relationships provide more useful feedback, because reviewers do not pay much attention to their language, words or phrasing and both developers are comfortable and direct with each other. However, being close to the author and having no such previous work experience with them seems to make the process harder. It seems difficult to criticise their code for the first time, and

it can also be hard to be objective in the feedback:

> *"If the colleague is a totally nice person, which I like a lot, then it is difficult to say that this is bad, what you did. That hurts me. Then one has to beat around the bush and say: one can do that here and there ... those are things that I do not like so much … if the author is a good friend, then it's difficult, and in the next time I ask somebody else to do it."* INT3

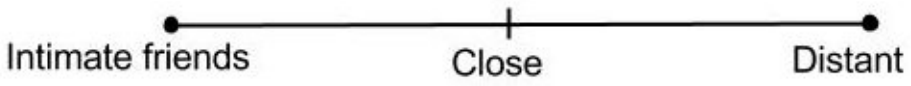

Fig. 4. Relationship Dimension

The interviews did not investigate the definition of friendship or close colleagues according to the interviewees, however, the interviews indicate that there might be a connection between the closeness of the developers and the level of the review professionalism. The level of closeness could be represented in a dimension according to Grounded Theory Methodology. The interviews indicate that if the developers were close enough (Fig. 4), then they would work with a sufficient level of professionalism and tend to be more direct and "honest" with each other. When the developers are either too close to each other – close friends – or to distant from each other, then they are more vulnerable and tend to be indirect with each other. This dimension is still an open question, since there is not enough data to form a conclusion. Further work can be directed by these two questions: First, when do developers consider themselves close friends and second, what is a good range of closeness in regard to the CR process.

3.2.5 Organisation

Not only the choice of language is dependent on the relationship, but also the manner of communicating. It is often easier to communicate face to face for many reasons. First, it is not necessary to express ideas in writing. Second, there is a smaller risk of miscommunication, as the author can hear the tone of the voice and perceive the body language of the reviewer. Last, the reviewer can see the author's immediate reaction to the feedback. Given all that,

it is no surprise that the developers prefer to meet face to face whenever possible rather than communicate via email or skype. Furthermore, working onsite saves time; it is more efficient to go directly to the colleague, to explain something to them and show them the related document, rather than sending emails back and forth or organizing a skype meeting. One interviewee explained:

> *"...Of course it is, I'd say, a bit easier, when the colleagues are onsite, to draw something fast or to look at certain documents, as if the colleagues are offsite, then you always have to do a Skype meeting, then you have to look that one has the data all on the computer and then explain, so you cannot just draw a sketch quickly or something like that."* INT1.

Moreover, working with colleagues onsite and having the ability to discuss suggestions with ease not only helps developers to know each other better and strengthens their relationship but also enables them to know more about each other's abilities and knowledge. As mentioned above, the reviewer considers the knowledge of the author while reviewing and writing the feedback. Thus, when the reviewer has a correct and precise insight into the author's knowledge, that helps them to write appropriate feedback. This is because the reviewer can better estimate how concise or detailed their feedback should be in order to deliver their message to the author. For example, when an author makes a mistake regarding a specific guideline, yet they have considered this guideline in previous codes, then the reviewer will know that it is just a lapse. The reviewer would not have to provide arguments explaining why this is wrong and would only point out the mistake, which saves time to invest in dealing with real mistakes. Otherwise, if the author is new to the project, and it is likely that they are not aware of all the guidelines, then it would be more reasonable to explain why they are mistaken and address the guideline or documentation concerned. In addition, knowing the author helps the reviewer, given the author's knowledge, to give them the right tips and the suitable explanations.

> *"I like to review the code of the fresher and the least experienced developers to make them aware of the practices we are following and also ensure they understand the comments that are coming from them"* INT2

Fig. 5. Relationship, Knowledge and Feedback

Furthermore, getting to know the abilities of the developers in a team could help organise the project. Given the abilities of the developers, the reviewer can better estimate how much time and effort is needed to review their code and how many rounds of CR are necessary before the code is ready to be checked out. That avoids planning and time problems, whereby the reviewer will have sufficient time to review the code and give useful feedback. One interviewee said while describing an unpleasant experience with CR:

"*there was no time to deeply review the code or to revise it*" *INT2*

Another said:

"*it takes more time in the case of low-quality code*" *P-INT3*.

Hypothesis (6): Better relationships provide a better perception of each other's knowledge, which leads to better process organisation and time management.

3.2.6 Results

The CR process has two different results, as far as learning is concerned. Either it is a successful experience, where the developers were satisfied because they learnt something or taught someone something, or they were dissatisfied and upset, and neither the author nor the reviewer has learnt anything out of this process and hence the process is considered unsuccessful. The results self-evidently impact the knowledge of the developers, as it expands when learning occurs. Significantly, it also influences the relationship between

the two developers. A developer tends to respect and trust the one who teaches them something and also the ones who learn "quickly"$_{INT1}$ and do not repeat their mistakes.

> *"I think I have more knowledge now and [feel] more confident when writing the comments. Before, I was new to the projects. Now I know the code and the project better, so only then, colleagues start respecting you, respecting your opinion. Because when you are fresher and new to the project, they think if he is new to the project, how can he give us the comments. And once they know – yeah – he also knows something, and we could use his comments constructively"*$_{INT2}$

Another developer said:

> *"So I'm happy when they find a 'real' mistake in my code; where I would say yes that's wrong. That could have been done better. When I see that, and I acknowledge that, then I revered them, because then I see, they have penetrated the problem and understood it. And they have found the mistake that I had made. Then I can finally rely on them, that the code is good"*$_{INT3}$.

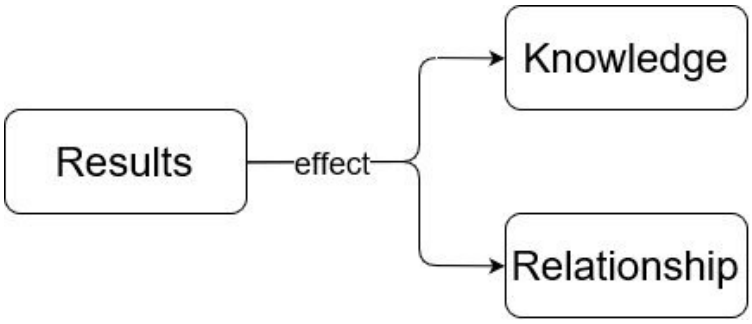

Fig. 6. Results of one CR on Relationship and Knowledge

Moreover, after a successful learning or teaching process, developers gain a better understanding of each other's knowledge and expertise. A. Bosu, J. C. Carver, C. Bird, J. Orbeck, C. Chockley [1] also conclude this in their research. Furthermore, after such a learning process, developers also become closer to each other and are more used to discussing and talking to each other, which facilitates future collaboration and makes the next CR process smoother. According to Baum et al. [13], the importance of negative social effects among developers decreases with time when reviews are in regular use.

Hypothesis (7): The results of one learning procedure influences the relationship between the reviewer and the author, the author's knowledge, and the way each of them perceives the knowledge of the other. As a consequence, it affects the upcoming code reviews.

Here (Fig. 7) are all the hypotheses represented in one model.

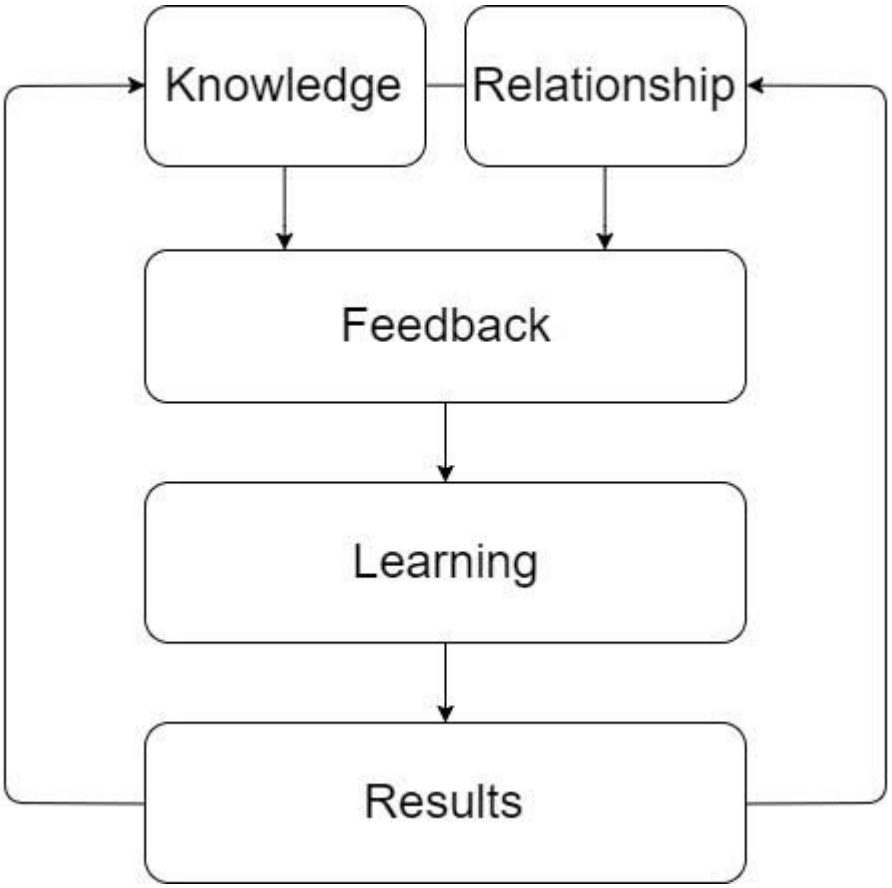

Fig. 7. Complete Model

3.3 Episodes

Here are two examples to understand the whole theory and connect the hypotheses with each other.

3.3.1 Positive Episode

Let us assume that developer A, wrote a code and asked developer B to check it for him. Developers A and B have worked together before and have checked each other's code. B has a good perception of A's knowledge and experience, because he has checked multiples of A's codes. Thus, B could estimate

how much time he needed for the review (Hypothesis 6). While B was reviewing, he knew, where he should pay most attention; on which aspects he should concentrate, since he knows where A's mistakes usually occur (Hypothesis 6). B found three main mistakes in the code.

- The first mistake was, given A's knowledge, a silly mistake. The reviewer B recognized that this is an oversight (Hypothesis 4), so he only points it out and does not take the time to demonstrate why it is wrong (Hypothesis 6).
- The second mistake A has made, is related to an old guideline. B remembered that A was not in the team yet when this guideline was added and maybe he is not aware of it, so B linked the guideline and explained briefly in the comments what A may have missed (Hypothesis 4). After that, A read the comment and learnt about the guideline (Hypothesis 3). A was happy and relieved, because he has caught up some of what he has missed so easily (Hypothesis 1) and he is more self confident (Hypothesis 2), and admired B, because he helped him (Hypothesis 2).
- The third mistake is the type of mistake A has been making frequently, even though B pointed it out in the last review and linked a tutorial that explained it. Therefore, B without hesitation asked A to meet, in order to understand what is wrong and to explain to him in person how to avoid this mistake. In the meeting, B was honest and direct with A (Hypothesis 5), he showed him the repeated mistakes and explained how they are related, then explained how he can correct and avoid them in future. A, similarly, asked B during the meeting for more examples and discussed with B alternative ways to avoid this type of mistake (Hypothesis 5), that B had not thought of before. After the meeting both developers learnt from each other and were happy and satisfied (Hypothesis 1).
  Both developers considered this process to be a success, not only because the mistakes were corrected but because they benefited from it as well (Hypothesis 1). After this process both developer's knowledge expanded, they got to know each other more and gained a better insight into each other's knowledge (Hypothesis 7).

3.3.2 Negative Episode

Developer C is a new junior developer in the team, he wrote a code and asked if someone could review his code for him. Developer D, a senior developer and one of the first members of the team had two hours free that day and

decided to check C's code. This was the first collaboration between them.

When D reviewed the code, he found many mistakes. He was disappointed by C's lack of knowledge and the fact that many guidelines were not considered, however he did not know that C was new to team nor that he is a junior developer. D wanted to tell C, that he should read the project documentation carefully and send him some tutorials, but he hesitated (Hypothesis 5). D commented on all the mistakes and briefly explained and demonstrated them (Hypothesis 4), therefore he spent more than two hours reviewing the code (Hypothesis 6). D sent the review comments to C. C was overloaded with comments, he knew where he had gone wrong in some cases but had made these mistakes because he was confused (Hypothesis 4), others he did not understand, though he did not ask D about them (Hypothesis 5). Therefore, C did not really learn from the feedback and was not mentored (Hypothesis 3), even though he was expecting to learn much from the CR process (Hypothesis 1). Unfortunately, developer D thought less of developer C and avoided reviewing his code in the future (Hypothesis 7). In addition, C was annoyed with D, because he was expecting to learn from him and therefore C and D never had a good relationship (Hypothesis 7).

4 Suggestions

In this section, some improvement suggestions are introduced, to improve the CR process. The suggestions are based on the grounded theory of this work.

4.1 Relationship

The relationship between the author and the reviewer plays an important role in the CR process and in the learning process. According to hypotheses 4 and 5, good relationships lead to more useful feedback and also helps organize the code review process. That implies, improving the relationship between the developers and increasing the awareness of its role in CR would smooth and improve the CR process.

1. **Estimate the current state of the relationship**
   Relationship, according to Hypothesis (5), plays a significant role in the CR process and in the learning process, both reviewer and author

should be aware of that. Thus, I suggest that the reviewer before writing the comments and sending the feedback should evaluate their relationship with the author, consider the knowledge of the author, Hypothesis (4), and given that evaluation they would better know how to handle this process. Following is an handout for the reviewer, to facilitate the process of estimation. There are also suggestions for the appropriate action regarding the corresponding status of relationship.

Table 2. Relationship Estimation

<table>
<tr><th rowspan="2">My relationship with the author</th><th colspan="2">Aspects to consider</th></tr>
<tr><th>Experience & Project Knowledge</th><th>Criticism Acceptance</th></tr>
<tr><td>I do not know the author</td><td colspan="2">You do not know anything about the author; not about their experience and knowledge nor about their reaction to criticism. This where your relationship starts, so start a positive relationship and get to know each other. It is recommended to have a small talk together before starting the process, where you discuss your experience and knowledge, and the feedback method and learning process.</td></tr>
<tr><td>We have worked together before</td><td rowspan="3">You should ask yourself: how much do you know about the author's experience regarding this task and how much time they spent on this project. According to these criteria you can start giving feedback and evaluate their work.</td><td>Code review contains criticism, which does not occur in other tasks and is not always easy to accept. Accepting criticism is a skill, thus, it is important to know to what degree the author masters it.</td></tr>
<tr><td>We are friends</td><td>It can be hard for the reviewer to criticise the code of a friend or to be very direct with them about the quality of their code. So as a reviewer, you should try to be abstract and address the code in the feedback.</td></tr>
<tr><td>We work offsite</td><td>Working offsite is risky, since it is easy for the author to misinterpret the content of the feedback and it is harder for the reviewer to assessthe author's acceptance of criticism, since they cannot see their immediate reaction to it. Hence, in such cases reviewer and author should be more careful and should use any chance of meeting, workshops for instance, to know each other better. It is also advisable in this case, to have skype meeting, when personal meetings are impossible.</td></tr>
</table>

Red on Fig.3 means that the reviewer should be very careful, green means that the risk of conflict is low, and yellow is in between.

Member Check: member Check or informant feedback is a internal validity technique in qualitative research. It is used to improve the accuracy, credibility of a study. I performed this technique on this handout. This table was shown to one developer and asked about their opinion. They found that it has important aspects and such a model would

be useful for the reviewers. They said that they would like to apply an improved version of it. They also agreed that offsite working is risky as the table showed, but is not related to relationship. They recommended separating this factor from kind of relationships and to deeply study it. Moreover, they suggested adding the factor of team-size, because they believe that there is a connection between relationships and the size of the team. Last, they agreed that the yellow and red fields are risky and reviewers should be at those cases more careful. However, they think that the green fields are not safe but rather natural.

2. **Reviewers and authors have to collaborate outside the reviews**
   It is recommended that the reviewers and authors have mutual work aside from the reviews [22]. So they will get the chance to work together in a relative easier atmosphere, where criticism is not the centre of the work. That will help both developers to maintain a level of professionalism and mutual respect and avoid having strained relationships [22], which influences learning according to Hypothesis (5).
3. **Enlighten the importance of positive feedback**
   Reviewers only focus on what is wrong with the code, but not on what is good in the code. However, reviewers should praise the work of the author, when they find something written in a particularly good way or when they learn something from the code. To prevent the author from seeing the review process as trolling process and avoid creating an unpleasant atmosphere between the developers, which lead to a bad relationship between them. Positive feedback, in addition, assures the author of the correctness of their work and increases their self-esteem and confidence, which lead to less mistakes in the next code. To summarize, reviews are a good opportunity to improve relationships, support positive behaviours and to motivate the developers. Reviewers should be aware of that and make sure to write positive feedback, when possible, in the reviews.
   Although, reviews take the form of comments in the code, and it would be unprofessional to give code to the customer, with a "well done" comment. Developers need a tool for the CR, which enables them, to separate the comments from the code but in the same time to refer the comments, to a specific line in the code.

4.2 Learning

Learning in CR not only improves the whole team and with it the quality of the code, but also as show in Hypothesis 1, it gratifies the team members and improves their self-esteem as well. Moreover, as Bacchelli and Bird [10] suggested, code review process should be improved by improving its unexpected outcomes; such as learning. Therefore, the following suggestions will be given to improve the learning process in code review.

1. **Set learning as a goal for code review**
   Learning should be included in the goals to be achieved from code review. Developers should be informed from the beginning, that one of the reasons CR is executed is to learn from each other. When this is clear for the authors it will be easier for them to accept criticism and consider this process as a learning opportunity. It will also motivate them to ask questions when the suggestions for change are not clearly reasoned and to conduct discussions. In addition, when reviewers know that learning is a reason for CR and not just an extra benefit of it, they will put in more time and effort to teach the author, to explain the change suggestions and will make sure that the goal is achieved.
   Many reviewers think that reviewing takes up too much of their time. Thus, it would be ideal to let them know how much time should be allocated to reviews, so they can recognize whether they are spending an appropriate amount of time on reviews or not. That will also let them have the time required to explain all the unclear or difficult issues that arose in the code. To conclude, it is important to set learning as a goal of CR and to assign time for it.
2. **General vs. individual Feedback**
   In the previous section,Hypothesis (4), the influence of relationships on the CR was described. Good relationships have a good influence on CR, but a close relationship can have a bad influence on CR as well. There are in general two types of reviews. The first type is general feedback, where the reviewer gives explanations for their change suggestions regardless of the identity of the author, in order to avoid the negative effects of relationship on the review process. The second type, on the other hand, is the risky one, where the reviewer writes individual feedback. In this type the reviewer considers the author's knowledge and abilities when writing the feedback, in order to spend their time

teaching and explaining the mistakes to the author, instead of wasting it writing unnecessary comments.

Taking the author's knowledge and experience into consideration while reviewing the code and writing the review, leads to better process organization and time management, according to Hypothesis (6), but can also lead to conflicts, as shown in Section 3.2.4. The best overall review method has not been found. Although, I think the fear of conflict is not a sufficient reason to choose one method rather than the other. Hence, solutions to the conflict should be found in order to benefit from individual feedback.

Here are some tips that could help the reviewer to avoid problems related to relationships. I suggest that the reviewer distinguishes between the two types of feedback:

1. The first type is code-related review, when the reviewer finds a piece of code that could be improved, or a guideline is not followed, or a mistake. Here the reviewer should make sure to address the code and not the author, they can avoid using "you", for instance. In order, to separate the mistakes from the author; in other words, to indicate that these mistakes are "mistakes in the code" and not the "author's mistakes".
2. The second type of feedback is addressed to the author to improve their skills, where the reviewer writes a comment to inform or teach them about something. For example, the reviewer notices that the author always makes the same type of mistakes, the reviewer tries in such a situation to explain these mistakes to the author and takes the role of a teacher or mentor. In this case the reviewer should be careful with their phrasing and explanation. They should write the comments in a positive tone, include links to useful sites where the authors can read more about the topic, and phrase comments in the form of a question, such as "what about…" or "how about…".

3. **All combinations are useful**

   One of the useful things about CR is the chance to see different perspectives and different coding styles. When only a particular group of developers with similar experience review each other's code, then no new knowledge is shared. According to Hypothesis (2), knowledge plays an important role in the CR process and also affects learning. Therefore, it is important in CR to have developers with different ex-

perience share their knowledge. Because every developer brings new knowledge to the team:

- Experienced developers have knowledge in different fields. They have specialized knowledge and experience to pass on to less-experienced developers.
- Developers who have spent more time in the project than other team members know more about the project, since they are more familiar with the documentation of the project, the guidelines, the requirements, the standards, etc. They can teach others, especially those new to the team, project related-knowledge.
- Newcomers to the team may carry new ideas from their previous tasks, which can be useful to the team.
- Junior developers may have limited experience, however, they may use the latest coding styles or know about new technologies. They can teach this up to date knowledge to the other team members. Sometimes the "fresh" view of a newcomer, helps the team to change perspective and question existing structures.

For this reason, every possible combination of developers in a team should participate in CR. Experienced developers should review the code of the less experienced and vice versa, juniors should review the more experienced, and so on.

4. **Exchange Roles**

   Developers can learn from the process of giving feedback, as shown in Hypothesis (1). By explaining a suggestion for change they reinforce their own understanding, for instance, and they can also learn when receiving feedback. They learn how to hold the guidelines, for example. Thus, when a developer takes only one role, either that of author or reviewer then they will lose the benefits of the other role.

   Furthermore, having both roles improves the relationships among the developers, because it brings them all to the same level. The developers of the teams, in which developers held both roles, such as team A, benefited more from the CR process. Because having both roles prevents hierarchies among the developers and leads to better relationships. In contrast, in team B, where developers always had only one role in the review process, there was a big difference between the knowledge of the authors and of the reviewers. For all the aforementioned reasons, it is reasonable for the developers to have both roles; author and review-

er.

5. **Define "Ready to review"**
   When developers know that their code will be reviewed by another developer, they may feel encouraged to do their best to create elegant code. On the other hand, that may also motivate them to neglect the quality of their code because it will be checked anyway by another developer, which impacts the reviewing time. In order to avoid receiving a code which puts a greater burden than necessary on the reviewer, the quality of the code sent to be reviewed should be determined. I suggest that the team determine a "ready to review" checklist where some of the prerequisites are written. Checklists are strongly recommended as a method to find the things one is expected to do [23]. In this way, the author would know what is expected from them and the reviewer will not waste their time on things they are not responsible for. Most important developers would have more time to spend on teaching and learning. Here are some of the requirements the reviewer expected to be met before reviewing the code:
   Does the code compile without errors and run without exceptions in happy path conditions? Or are there coverable appropriate tests which can be carried out? Does the code pass these tests?
6. **Encourage meeting-based review**
   Meeting-based reviews cost significantly more than non-meeting-based reviews and do not find many more defects than the other type of reviews [4], however, conducting them from time to time has a positive impact on relationships and consequently produces more effective feedback, in accordance with Hypothesis (3). As mentioned in Section 4.1, it is challenging to communicate issues without personal meetings. Meeting in person or over video chat reduces the probability of misinterpretation between the reviewer and the author, because of the tone of the voice, the body language, and the spontaneous reaction. It also provides the chance for the developers to have small talks and get to know each other. Moreover, discussing the issues and explaining them in a constructive and professional way is much easier when people are talking face to face. That is why I suggest meeting-based reviews from time to time, and encourage the developer to discuss their knowledge when meeting for the first time.
   In meeting-based reviews, the author should prepare their code before the review, this preparation has two advantages. First, the author re-

thinks the reasons and methods behind each code modification which help them to understand their own code better. Second, the author double-checks their code and may find some of the defects before the review even begins. According to IBM [24], reviews with author preparation have much less defects than the ones without. In that way the meeting time is spent on the things that author really did not find or know, which makes the review more efficient.

7. **Gradually improve the code**
   Sometime the author writes a low-quality code and the reviewer tries to develop it into a super high-quality code. The reviewer may write too many change suggestions, so that the author will be lost in the hundred comments and cannot distinguish between the things they must change and what they could change. It would be more reasonable, to send these changes over more than one review round, so the author will not be overwhelmed and would have time to really comprehend every change suggestion. The reviewer can start with the things that have to be fixed and end it with the "to do" changes. However, if these changes were to need a lot of review rounds, then it is recommended to stop when the quality of the code is acceptable but yet not perfect, to prevent draining the author's patience. In the end, the developer cannot learn everything in one round. The reviewer can approximately determine when the code has "poor-quality", "acceptable-quality", or "high-quality" *see table x maybe*. After every review the quality of the code must be improved in the direction of "high-quality". The review rounds should be allowed to end when the code has at least acceptable quality. "Poor-quality" and "high-quality" are relative to the experience of the reviewer but "acceptable-quality" must be predefined. In conclusion, the reviewer has to improve the code and teach the author gradually, starting with the necessary changes and then making less important suggestions.

5 Related Work

There are previous studies that have examined the effects of different factors on the code review process. Most of them examined the heavyweight, intensive software inspection techniques, but only few of them have examined the lightweight code review procedures [25].

Baysal et al. found a number of factors; such as review size, reviewer characteristics, or author experience, to have significant impact on code review response time and outcome [12]. A. Porter, H. Siy, A. Mockus, and L. Votta [26] also investigated the effect of variance between elements of the software inspection process such as, the size of the team and the number and order of sessions, on the effectiveness of the inspection. In the study of Baysal et al. [12], they found that the reviewer activity, organization, component and patch size affect the response time. Weissgerber et al. [27] studied patch contributions by executing data mining on the email archives of two projects. They also found that the size of the patch affects the decision to accept the patch into the codebase. Jiang et al. [28] found that developer experience, patch maturity, and priori subsystem churn also affect the probability of the acceptance. Moreover, they found that, the submission time, the number of affected subsystems, and the number of suggested reviewers and developer experience impact the reviewing time. Furthermore, I am not aware of any research, which has studied the impact of relationship on the process of CR except for the research of A. Bosu, J. C. Carver, C. Bird, J. Orbeck, C. Chockley [1]. They found that in addition to the reputation of the author and area of expertise, relationship with the author and previous interaction with them affect the probability of the acceptance. They also found that a high quality code impacts impressions regarding the author, relationship and future collaboration, which assure the Hypothesis (6) of this research. However, all of the aforementioned studies have focused on the factors effect on accepting and rejecting the author's review request, or on the outcomes of the process. While this study focused on how some of these factors; for instance, developer knowledge or relationship, have affected the process itself; which mean the process of reviewing the code and giving the feedback.

Furthermore, I am not aware of any research, that studied the impact of relationship on CR process with the exception of the research of A. Bosu, J. C. Carver, C. Bird, J. Orbeck, C. Chockley [1]. Where they found that in addition to the reputation of the author and area of expertise, the relationship with the author and previous interaction with them affect the probability of the acceptance. They also found that high quality code has impact on the impressions about the author, relationship and future collaboration, which assure the Hypothesis (6) of this research. All of the aforementioned studies have focused on the effect of these factors on the acceptance and rejection of the request of the author to the reviewer, or on the outcome of this process. While this study

focused on how some of these factors; for instance, developer knowledge or relationship, have affected the stage of reviewing and giving the feedback.

Furthermore, most of the studies on modern code reviews have examined the code review process and factors that affect review interval or outcome, I am not aware of any study that has explored the factors influencing the learning outcome of code review. However, Latoza et al. [29] highlight the importance of knowledge transfer because they found, while studying developer work habits, that many problems encountered by developers were related to understanding the rationale behind code changes and gathering knowledge from other members of their team. And Bacchelli and Bird while examining the purposes and outcomes of code reviews also found that code reviews offer additional benefits such as knowledge sharing and identifying better solutions and they also recommended embracing these unexpected outcomes of CR.

6 Limitation

Since qualitative interviews were chosen as the source of data, the amount of data was large compared to the time available, but still not sufficient to make generalisations. Completed theoretical sampling means to continue collecting data or, in this case, interviewing until new data stops emerging, which in this research could not be done, due to lack of available time. In this research 7 interviews – 3 of which are pilot interviews – were conducted over three weeks. Additionally, considering other sources such as observing multiple CR processes or reading the exchanged comments, then comparing them with the current data could provide more reliable results. Hence, the reader should keep in mind that the results are based only on the interviews; the testimony of the interviewees. Moreover, only authors and reviewers were interviewed. Further research could include interviewing other team members that participate in one way or another in the CR process such as, team-lead or Chief Architect. In addition, the results represent only one company further research could include different companies.
The quality of a grounded theory can be assessed based on four categories: Credibility, originality, resonance and usefulness, according to Charmaz [30]. Regarding originality, comparison was made between this work and other similar studies in Section 5; related work. Regarding usefulness, in Section 4; suggestions and proposals based on the theory are suggested to the practitioners in the CR process to improve their process and its outcome. As for credibility,

some procedures were carried out, but they are not sufficient. There is a risk of bias during the analysis of the qualitative data, for instance, when important data is not taken into account or open questions are not pursued. In this study, this risk is reduced due to the use of grounded theory practices: coding, constant comparison, theoretical sampling and memos. At the end of each interview, the interviewee was asked to raise any points they felt may have not been covered in the discussion. However, no complete validation procedure was conducted on the theory.

7 Conclusion

This study was performed to examine the CR processes in some teams at *Capgemini* and in the end offer some suggestions to improve them. The research is based on GTM. Seven interviews were conducted from four different teams. After analysing these interviews, the six following hypotheses were generated:

- Hypothesis (1): When developers learn something from the code review process, then they are satisfied with it, independently of whether they faced conflicts during the process or not.
- Hypothesis (2): Due to self-estimation during the learning process (by correcting or being corrected through others) the confidence of the team members increases.
- Hypothesis (3): Learning from the code review is dependent on the feedback.
- Hypothesis (4): Author's knowledge affects the feedback of the code review.
- Hypothesis (5): Good relationships lead to more useful feedback.

This finding warrants further research in two directions: First, definition of friendship and second, the perfect relationship in respect to the feedback.

- Hypothesis (6): Better relationships provide a better perception of each other's knowledge, which leads to better process organisation and time management.
- Hypothesis (7): The results of one learning procedure influences the relationship between the reviewer and the author, the author's knowledge, and the way each of them perceives the knowledge of the other. As a conse-

quence, it affects the upcoming code reviews.

To conclude, learning in the CR process is dependent on feedback. The feedback is affected by two factors: the author's knowledge and the relationship among the developers. Additionally, better relationships among the developers provide a better perception of each other's knowledge, which in turn influences the organization and planning of the CR process. When developers learn from CR, they consider the process successful and are satisfied with it. Every CR process between two developers influences the upcoming CR process between them, as the results of learning affect the relationship, knowledge, and perception of each other's knowledge.

Moreover, based on these hypotheses multiple suggestions to improve the CR processes in the different teams were given. These include:

- First, the reviewer and the author have to estimate the current state of the relationship before beginning the process in order to be aware of the difficulties that they may face.
- Second, it is important to highlight the role of learning and knowledge-sharing in the team.
- Third, meeting-based reviews are recommended.
- Fourth, any combination of two developers should do the code review together.

This work also expressed the need of a code review tool, which would further support communication between the author and the reviewer.

A Appendices

A.1 Interview Questionnaire (German Version)

Reviewer:

Icebreaker Question

Erzähl mir vom Code Review Prozess in deinem Team.

Code Review Style

1. Welche Aspekte betrachtest du zuerst beim Code Review?
2. Wie erkennst du einen guten Code?
3. Wie helfen Code Reviews dazu, einen Code mit niedriger Qualität zu verbessern?

Persönliche Aspekte

1. Erzähl mir von jedem Mitglied des Code Review Prozesses.
2. Um über die Beziehungen zwischen den Reviewer und Entwickler zu erfahren.
3. Gibt es einen Developer/Entwickler, dessen Code du gerne prüfst/reviewst? Erzähl mir von eurem letzten Code Review.
4. Erinnerst du dich an die Ersten Code Reviews in deinem Team? Erzähl mir von einem davon? Was war am Anfang besonders schwierig?
5. Die Idee dahinter ist zu wissen: ob die Beziehung zwischen den Mitarbeitern eine Rolle spielt (z.B. die Ehrlichkeit) und ob die Beziehung sich entwickelt während der Zeit.
6. Was nutzt ein dir Code Review?

Konflikte

1. Erzähl mir von einem erfolglosen Code Review.
   Um zu wissen:
   - Wann ist ein Code Review erfolgreich.
   - Was kann ein Code Review hindern.
   - Was ärgert die Reviewern an Entwicklern und andersrum.
2. Erzähl mir von deinem letzten Änderungswunsch, der nicht akzeptiert wurde?
3. Wie gehst du mit Konflikte beim Formulieren dein Code Review? (dann: Erzähl mir von deinem letzten Code Review; wie hast du die Änderungswünsche formuliert)
4. Nur Wenn Zeit übrigbleibt: Wie unterscheiden Test und Code Review sich in der Zusammenarbeit?

Author:

Icebreaker question

Erzähl mir vom Code Review Prozess in deinem Team.

Code Review Style

1. Was erwartest du von Code Review?
2. Wie helfen Code Reviews dazu, einen Code zu verbessern?
3. Was nutzt ein dir Code Review?

Persönliche Aspekte

1. Erzähl mir von jedem Mitglied des Code Review Prozesses
2. Um über die Beziehungen zwischen den Reviewer und Entwickler zu erfahren.
3. Gibt es einen Reviewer, von dem du deinen Code gerne prüfen/reviewen lässt? Erzähl mir von eurem letzten Code Review.
4. Erinnerst du dich an die Ersten Code Reviews in deinem Team? Erzähl mir von einem davon? Was war am Anfang besonders schwierig?
5. Die Idee dahinter ist zu wissen: ob die Beziehung zwischen den Mitarbeitern eine Rolle spielt (z.B. die Ehrlichkeit) und ob die Beziehung sich entwickelt während der Zeit.

Konflikte

1. Erzähl mir von einem erfolglosen Code Review.
2. Wonach entscheidest du, ob du die Review-/Änderungs- anfrage akzeptieren oder nicht? Wem und wie kommunizierst du das?

A.2 Interview Questionnaire (English Version)

Reviewer:

Icebreaker Question

Would you tell me about Code Review Process in your team?

Code Review Style

1. What are the first things that you review in a code review?
2. How do you recognize a good code?
3. How do code reviews help improve a low-quality code?

Personal aspects

1. Would you tell me about every team member that participates in the Code Review Process?
2. Is there a developer whose code you like to review? Would you tell me about your last code review experience with them?
3. Do you remember the first code reviews in your team? Would you tell me about one of them? What was especially difficult in the beginning?
4. What are the advantages that CR would bring to you?

Conflicts

1. Would you tell me about an unsuccessful code review?
2. Would you tell me about your last change suggestion which was not accepted?
3. How do you handle conflicts when formulating your feedback? Would you tell me about your last code review? How did you formulate the change suggestions?
4. Only when there is time left: How do test and code review differ as far as collaboration is concerned?

Author:

Icebreaker Question

Would you tell me about Code Review Process in your team?

Code review style

1. What do you expect from Code review?
2. How do code reviews help improve a low-quality code?
3. What is the advantage CR would bring to you?

Personal Aspects

1. Would you tell me about every team member that participates in the Code Review Process?
2. Is there a reviewer by whom you want your code to be reviewed? Tell me about your last code review experience with them?
3. Do you remember the first code reviews in your team? Would you tell me about one of them? What was especially difficult in the beginning?

Conflicts

1. Would you tell me about an unsuccessful code review?
2. How do you decide if you will accept a change suggestion or not? How do you communicate this and with whom?

Bibliography

[1] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, 2017.

[2] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976 [Online]. Available: https://doi.org/10.1147/sj.153.0182

[3] B. Bruegge and A. H. Dutoit, *Object-oriented software engineering using uml, patterns, and java*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[4] L. G. Votta Jr., "Does every inspection need a meeting?," *SIGSOFT Softw. Eng. Notes*, vol. 18, no. 5, pp. 107–114, Dec. 1993 [Online]. Available: http://doi.acm.org/10.1145/167049.167070

[5] SmartBear, "Best practices for code review" [Online]. Available: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/ [Accessed: 2017-12-01]

[6] J. A. Cohen, *Best kept secrets of peer code review*. Printing Systems, 2006 [Online]. Available: https://books.google.de/books?id=b9ywHanWN5kC

[7] M. E. Fagan, "Advances in software inspections," *IEEE Transactions on Software Engineering*, vol. 12, no. 7, pp. 744–751, 1986.

[8] E. P. Doolan, "Experience with fagan's inspection method," *Softw. Pract. Exper.*, vol. 22, no. 2, pp. 173–182, Feb. 1992 [Online]. Available: http://dx.doi.org/10.1002/spe.4380220205

[9] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," in *2013 ACM / IEEE international symposium on empirical software engineering and measurement, baltimore, maryland, usa, october 10-11, 2013*, 2013, pp. 133–142 [Online]. Available: https://doi.org/10.1109/ESEM.2013.23

[10] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th international conference on software engineer-*

ing (icse), 2013, pp. 712–721.

[11] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1028–1038 [Online]. Available: http://doi.acm.org/10.1145/2884781.2884840

[12] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review.," in *WCRE*, 2013, pp. 122–131 [Online]. Available: http://dblp.uni-trier.de/db/conf/wcre/wcre2013.html#BaysalKHG13

[13] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "Factors influencing code review processes in industry," in *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*, 2016, pp. 85–96 [Online]. Available: http://doi.acm.org/10.1145/2950290.2950323

[14] A. Strauss and B. Glaser, *Awareness of dying*. Aldine Publishing Company, 1965, 1998.

[15] A. Strauss and J. M. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory*. SAGE Publications, 1998 [Online]. Available: https://books.google.de/books?id=wTwYUnHYsmMC

[16] H. A. Stewart, "Teories-in-use and espoused theories: An examination of team decision-making in the initial special education eligibility meeting," 2015 [Online]. Available: http://trace.tennessee.edu/cgi/viewcontent.cgi?article=4761&context=utk_graddiss

[17] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*, 3rd ed. Sage Publications Ltd., 2008.

[18] P. Mayring, *Qualitative inhaltsanalyse : Grundlagen und techniken*, 12., Überarb. Aufl. Weinheim \[u.a.\]: Beltz, 2015 [Online]. Available: http://scans.hebis.de/HEBCGI/show.pl?36415386_kla.pdf

[19] B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*. New York, NY: Aldine de Gruyter, 1967.

[20] "State of code review 2017: Trends and insights into dev collaboration," *Collaborator*, 2017 [Online]. Available: https://smartbear.com/resources/ebooks/the-state-of-code-review-2017/. [Accessed: 05-Apr-2017]

[21] AIFORSE-Community, "What's wrong with software engineers commu-

nication?" [Online]. Available: https://medium.com/ai-for-software-engineering/whats-wrong-with-software-engineers-communication-62f609d4ce0e [Accessed: 2017-12-01]

[22] *Peer reviews in software: A practical guide*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[23] SmartBear, "Creating your code review checklist." https://blog.smartbear.com/development/creating-your-code-review-checklist/ [Accessed: 2017-12-01].

[24] IBM, "11 proven practices for more effective, efficient peer code review," January, 2011 [Online]. Available: https://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/ [Accessed: 2017-12-01]

[25] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th working conference on mining software repositories*, 2015, pp. 146–156 [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820538

[26] A. Porter, H. Siy, A. Mockus, and L. Votta, "Understanding the sources of variation in software inspections," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 1, pp. 41–79, Jan. 1998 [Online]. Available: http://doi.acm.org/10.1145/268411.268421

[27] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!," in *Proceedings of the 2008 international working conference on mining software repositories, MSR 2008 (co-located with icse), leipzig, germany, may 10-11, 2008, proceedings*, 2008, pp. 67–76 [Online]. Available: http://doi.acm.org/10.1145/1370750.1370767

[28] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? And how fast?: Case study on the linux kernel," in *Proceedings of the 10th working conference on mining software repositories*, 2013, pp. 101–110 [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487111

[29] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proceedings of the 28th international conference on software engineering*, 2006, pp. 492–501 [Online]. Available: http://doi.acm.org/10.1145/1134285.1134355

[30] K. Charmaz, “Constructing grounded theory: A practical guide through qualitative analysis,” 2006.